Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Circuit Simulator Project Report

Author:
SOS101:
Yuexin Ding (CID:01856648)
Yuhong Ruan (CID:01869206)
Mengyuan Yin (CID:01842827)

Word Count: 10959
June 2021

Content Page

1 Introduction

This project aims to make a circuit simulator that reads the component netlist of a circuit and then produce magnitude and phase response by AC and DC analysis.

This report presents the decomposition of problems including assumptions made, requirements, design strategies and methods for DC and AC analysis. How each function is implemented is represented in the form of flowcharts which provide the working process of these algorithms. Tests for individual functions are listed along with the obtained results; any problems met and how solutions are found are explained in detail. Improvements are also included in this part. Tables and Gantt chart are used to present how this project has been planned and managed. The processes of several tests for the completed circuit simulator are shown, the results are compared with calculations done by professional software. Finally, recommendations will be made about how this simulator could be improved to further meet the needs of the users.

2 Design - Software Requirement Specification

2.1 Introduction

2.1.1 Purpose

The purpose of our project is to build a circuit simulator that performs DC and AC analysis for a given circuit and produces magnitude and phase plots as outputs.

2.1.2 Intended Audience

The intended audience of this software package is students who need to analyse electrical circuits. This program can help them to find the transfer function plots which would useful when analysing and designing circuits.

2.1.3 Project Scope

Our project is based on circuits containing linear devices: resistors, inductors, and capacitors; and non-linear devices: BJTs, MOSFETs, and Diodes.

2.1.4 Conventions and Acronyms

V_A	Ealy Voltage
I_s	Saturation Current
β_f	Forward Gain
β_r	Reverse Gain
k_p	Transconductance of PMOS
k_n	Transconductance of NMOS
K	K coefficient of CMOS
W	Width
L	Length
V_t	Threshold Voltage

Table 1: Table for acronyms used in this report

2.2 Overall Description

2.2.1 Product Overview

Our circuit simulator is a C++ software package that performs DC and AC analysis for a given circuit. The circuit will be passed into the program in the form of a netlist. DC operation is performed first to find the DC operating point, which is then used to calculate parameters of non-linear devices. Then AC analysis is performed based on these parameters. It will find the transfer functions of selected nodes and output a netlist that contains the magnitude and phase of a transfer function under different frequencies. Then MATLAB is used to plot a magnitude-frequency graph and a phase-frequency graph.

2.2.2 Assumptions and Dependencies

- For NPN BJT, we assume the type is 2N2222 which is a common one used before. The parameters for it are obtained from LTSpice and datasheet [1]: $I_s = 10^{-14}A$, $V_A = 100V$, $\beta_f = 200$, $\beta_r = 3$, $V_t = 0.025V$.
- For PNP BJT-we assume the type is 2N2907 which is a common one used before. The parameters for it are obtained from LTSpice and datasheet [2]: $I_s = 10^{-14}A$, $V_A = 120V$, $\beta_f = 200$, $\beta_r = 3$, $V_t = 0.025V$.
- N-MOSFET-parameters are obtained from LTSpice: $K = \frac{k_n}{2} \cdot \frac{W}{L} = 0.0025$, $V_A = 100V$, $V_t = 2V$
- P-MOSFET-parameters are obtained from LTSpice: $K = \frac{k_p}{2} \cdot \frac{W}{L} = 0.0025$, $V_A = 100V$, $V_t = -2V$.

Temperature: assume temperature does not affect the values and performance of components.

2.3 Features & Requirements

2.3.1 Functional Requirements

- The program should be able to read a netlist file and parse the data correctly.
- The program should be able to convert the components into their equivalent models to be used in AC and DC analysis
- The program should be able to perform DC analysis to find the operating point.
- The program should be able to perform AC analysis to find the transfer function.
- The program should be able to select the correct reference and output nodes. The output should contain a magnitude-frequency graph and a phase-frequency graph.

2.3.2 Non-functional Requirements

- Performance: the execution time of the program should be as short as possible. (The initial guess for Newton Raphson method should be chosen carefully to minimize the number of iterations, hence shorten the execution time)
- Portability and compatibility: The program should be portable, so moving from one OS to another OS doesn't create any problem.

2.3.3 System Requirements

C++ compiler: the code will be written in C++, so a C++ compiler is needed.

Eigen library for C++ (release version: Eigen 3.4-rc1): this is an external library that needs to be downloaded. It is used for the operation of matrices [3]. (Eigen doesn't have any dependencies other than the C++ standard library.)

The reason why the Eigen library is chosen is that it can manipulate matrix in complex form and solve matrix system equations quickly and accurately. In AC analysis, the conductance of inductors and capacitors, as well as the values of voltage and current sources are complex numbers. Hence, we need a tool to solve the matrix in complex form. Eigen library can fully meet our requirements. Thus, it is used as an external library to assist our program.

MATLAB: MATLAB allows the plotting of functions and data. It is used to plot magnitude-frequency and phase-frequency graphs.

2.4 Problem Decomposition

It is more efficient to write code that in functions and modules, because it will not only save storage space, but also become easier to both follow and maintain the code.

Therefore, we have decomposed the task into subtasks, which will be written in separate C++ files and their header files will be included in the main C++ file. Below is a breakdown/outline of all the tasks:

- Read and parse the netlist file (Section 4.3)
- Perform DC analysis to find the operating point
    - Set up the simulation:
        - Replace AC voltage sources and inductors with short circuits (Section 4.4)
        - Replace AC current sources and capacitors with open circuits (Section 4.5)
        - Change the non-linear components to their equivalents (Section 4.6)
    - Build the initial solution of a conductance matrix (Section 4.8)
    - Build the conductance matrix (Jacobian matrix) (Section 4.9)
    - Build the voltage and current source column (Section 4.10)
    - Use the iterative Newton-Raphson method (Section 4.11)
- Perform AC analysis to find the transfer function
    - Set up the simulation:
        - Replace DC voltage sources with short circuits (Section 4.4)
        - (DC current sources will naturally become open circuits)
        - Change the non-linear components to their equivalents (Section 4.6)
    - Build the conductance matrix (Section 4.12)
    - Build the voltage and current source column (Section 4.12)
- Output (Section 4.13)
    - Ask the user for the reference and output nodes
    - Write the results into a file
    - Use MATLAB to plot diagrams

2.5 Overall design

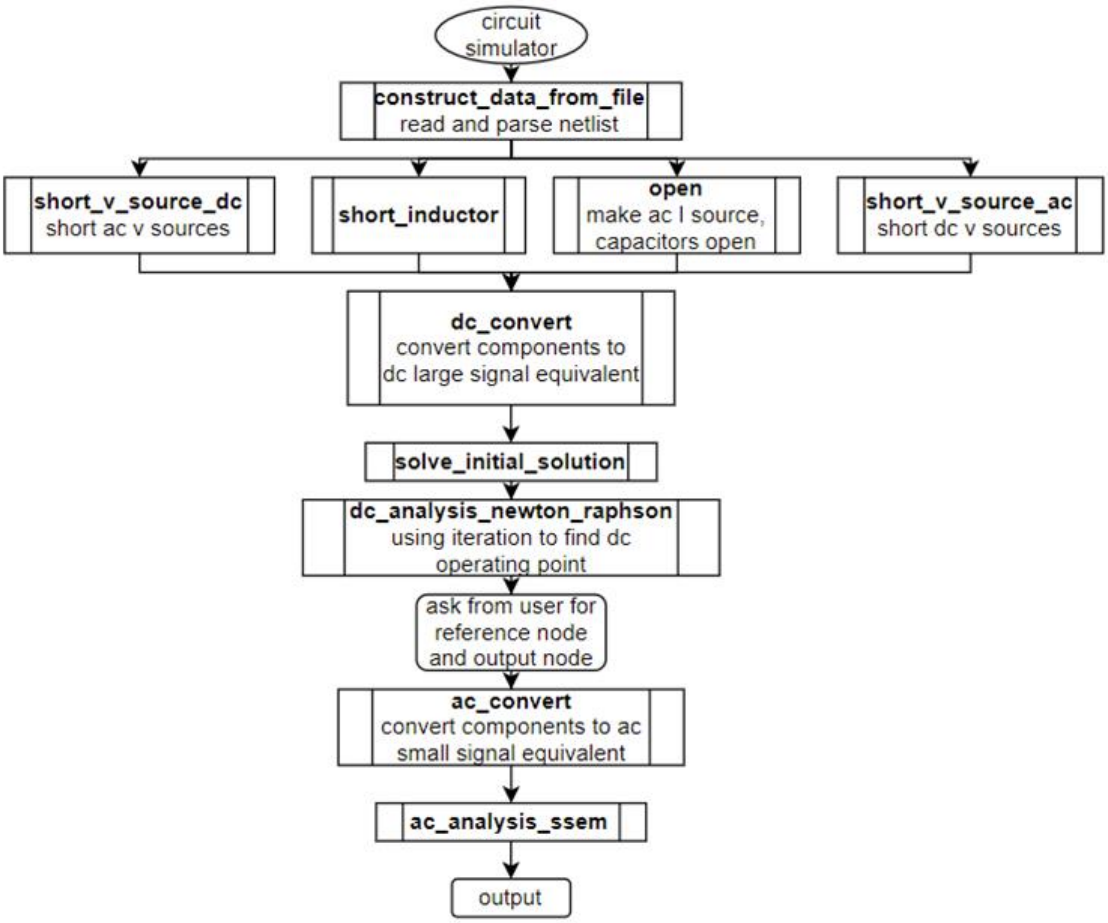


Figure 1: Overview of project

3 Principle and Analysis

3.1 Modified Nodal Analysis

Modified Nodal Analysis (MNA) can be used to analyze circuits comprised of linear components (i.e., resistors, capacitors and inductors), current sources and voltage sources. The main idea of MNA is writing the Kirchhoff's Current Law (KCL) equation for each node in matrix form and add on the equations representing voltage relations for terminals of voltage sources [4]. Then the matrix equations system is solved to obtain voltage at each node. The matrix equations system has the form $\boldsymbol{Ax} = \boldsymbol{b}$.

For a circuit with n nodes and m voltage sources, $\boldsymbol{A}$ is a matrix of size $(n+m) \times (n+m)$. It is divided into 4 parts.

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{G} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{0} \end{pmatrix}$$

$\boldsymbol{G}$ $(n \times n)$ is the conductance matrix. $G = \begin{pmatrix} g_{11} & -g_{12} & \cdots & -g_{1n} \\ -g_{21} & g_{22} & \cdots & -g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -g_{n1} & -g_{n2} & \cdots & g_{nn} \end{pmatrix}$. For $G(i,j)$,when $i \neq j$, we enter the negative value of the conductance connected to node i and node j. For the diagonal of $\boldsymbol{G}(i,i)$, we enter total conductance of components with one end connected to node i.

$\boldsymbol{B}$ $(n \times m)$ is made up of numbers 0, 1 and -1. Every voltage source corresponds to a new column. For the first voltage source, if node i is connected to the negative terminal of a voltage source, then $\boldsymbol{B}(i,1)$ is -1; if node j is connected to the positive terminal of the same voltage source, then $\boldsymbol{B}(j,1)$ is 1. Other entries in column 1 of $\boldsymbol{B}$ is 0.

$\boldsymbol{C}$ $(m \times n)$ is the transpose matrix of $\boldsymbol{B}$.

$\boldsymbol{x}$ is a vector of size $(n+m) \times 1$. The first n rows of $\boldsymbol{x}$ comprise of voltage at each node of the circuit. The last m of $\boldsymbol{x}$ represent currents passing through the voltage sources.

$$\boldsymbol{x} = \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_n \\ I_{V_1} \\ \vdots \\ I_{V_m} \end{pmatrix}$$

$\boldsymbol{b}$ is a vector of size $(n+m) \times 1$. For the first n rows of $\boldsymbol{b}$, $\boldsymbol{b}(i,1)$ is equal to the total current from current sources entering node i. The last m rows of $\boldsymbol{b}$ is made up of values of independent voltage sources.

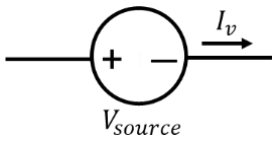


Figure 2: Illustration of current through voltage source

$$\boldsymbol{b} = \begin{pmatrix} I_{source_1} \\ I_{source_2} \\ \vdots \\ I_{source_n} \\ V_{source_1} \\ \vdots \\ V_{source_m} \end{pmatrix}$$

For example: -

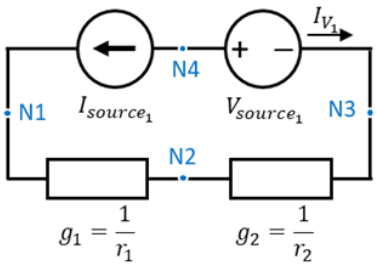


Figure 3: Example for AC analysis

The circuit can be expressed in matrix form:

$$\boldsymbol{Ax = b}$$

$$\begin{array}{c} \\ N1 \\ N2 \\ N3 \\ N4 \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} G & N1 & N2 & N3 & N4 \end{array} \\ \begin{pmatrix} g_1 & -g_1 & 0 & 0 & 0 \\ -g_1 & g_1+g_2 & -g_2 & 0 & 0 \\ 0 & -g_2 & g_2 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 \end{pmatrix} \\ \begin{array}{ccccc} & & & & B \end{array} \end{array} \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ I_{V_1} \end{pmatrix} = \begin{pmatrix} I_{source_1} \\ 0 \\ 0 \\ -I_{source_1} \\ V_{source_1} \end{pmatrix}$$

C

, which represents KCL equations:

$$N1: \frac{V_1}{r_1} - \frac{V_2}{r_1} - I_{source_1} = 0$$

$$N2: \frac{V_2}{r_1} - \frac{V_1}{r_1} + \frac{V_2}{r_2} - \frac{V_3}{r_3} = 0$$

$$N3: \frac{V_3}{r_3} - \frac{V_2}{r_2} = 0$$

$$N4: I_{V_1} + I_{source_1} = 0$$

, and relations of voltages at terminals of voltage sources:

$$V_4 - V_3 = V_{source_1}$$

MNA method is useful in both DC sand AC analysis.

3.2 DC Analysis

In DC analysis, we treat AC voltage sources as short circuits, and AC current sources as open circuits. Moreover, since when $\omega = 0$, capacitors are equivalent to open circuits, and inductors are equivalent to short circuits.

In DC analysis, we aim to find the DC operating point for non-linear devices, so that we can use the value for the AC analysis later on. This can be done using the Newton-Raphson method.

Newton-Raphson method is a well-known numerical method due to its simplicity in formula and fast convergence. It can approximate a solution to an equation. The root of the equation $f(x) = 0$ can be obtained by starting with an initial guess x_0 and iterating Equation 1.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (1.)$$

We can obtain improved approximation, until x_{n+1} achieves desired degree of accuracy. Equation 1 shown above can be extended to solve multi-dimensional equation system $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}$. In the case of circuit analysis, $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$, which is a vector containing voltages of all nodes in the circuit; $\boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}$, which is a vector containing KCL equations $f(x_i)$ at all nodes in the circuit. The approximating equation is given by:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - Jr(\boldsymbol{x}_n)^{-\boldsymbol{1}} \cdot \boldsymbol{F}(\boldsymbol{x}_n) \tag{2}$$

, where $Jr\ (\boldsymbol{x}_n) = \frac{d}{d\boldsymbol{x}} \boldsymbol{F}(\boldsymbol{x}_n) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$, which is called Jacobian matrix [5].

By multiplying both sides of the equation by $Jr\ (\boldsymbol{x}_n)$, we can obtain

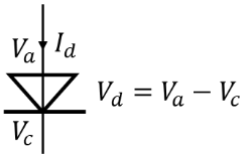
$$\boldsymbol{x}_{n+1} \cdot Jr(\boldsymbol{x}_n) = Jr(\boldsymbol{x}_n) \cdot \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_n)$$
$$\boldsymbol{x}_{n+1} = Jr(\boldsymbol{x}_n)^{-\boldsymbol{1}} \cdot (Jr(\boldsymbol{x}_n) \cdot \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_n)) \tag{3}$$

A correlation can be drawn from this Equation 3 and MNA analysis. $\boldsymbol{Jr}(\boldsymbol{x}_n)$ can be viewed as the conductance matrix. $Jr(\boldsymbol{x}_n) \cdot \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_n)$ can be treated as the RHS column, each entry represents the total value of current sources flowing into each node [6].

In order to solve Equation 3, we need to obtain the Jacobian matrix $Jr\ (\boldsymbol{x}_n)$ and $Jr(\boldsymbol{x}_n) \cdot \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_n)$ for the circuit. For resistors, $\frac{\partial f_i}{\partial x_i}$ is simply equal to its conductance, and $\frac{\partial f_i}{\partial x_i} \cdot x_i - f_i$ is equal to the total value of current sources flowing into node i. However, for non-linear components, the value of delta $\frac{\partial f_i}{\partial x_i}$, and $\frac{\partial f_i}{\partial x_i} \cdot x_i - f_i$ needs further investigation. This is elaborated below.

3.2.1 Diode

When diode is on, the current through diode is defined by Shockley's approximation [7].



$$I_d = I_s \cdot \left(e^{\frac{V_d}{V_T}} - 1 \right) \tag{4}$$

By partial differentiation, we can obtain:

$$\frac{\partial I_d}{\partial V_d} = g_d = \frac{I_s}{V_T} \cdot e^{\frac{V_d}{V_T}}$$
$$\frac{\partial I_d}{\partial V_d} \cdot V_d - I_d = -I_{source} = \frac{I_s}{V_T} \cdot e^{\frac{V_d}{V_T}} \cdot V_d - I_s \cdot \left(e^{\frac{V_d}{V_T}} - 1 \right) \tag{5}$$

The comparison between Equation set 5 and Equation set 3 proposes an equivalent model for diodes:

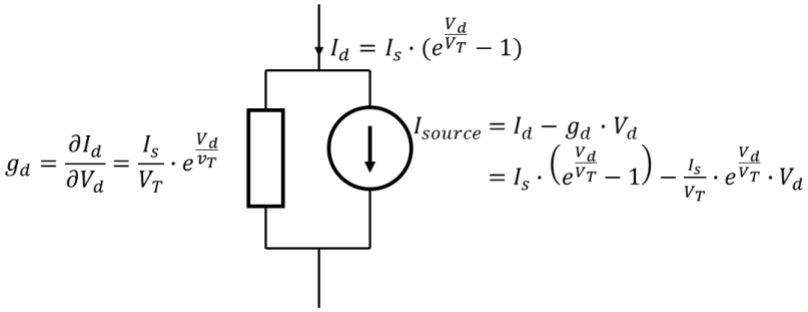


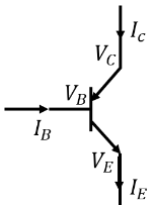
Figure 4: Equivalent model for diode

When diode is off, it is equivalent to open circuit, the conductance g_d and I_{source} will all be equal to 0.

3.2.2 Bipolar Junction Transistor (BJT)

A BJT has 2 diode-like junctions, namely Emitter-Base Junction (EBJ) and Collector-Base Junction (CBJ). Different combinations of forward-bias and reverse-bias mode of CBJ and EBJ makes four different modes of BJT. The following explanation takes NPN BJT as an example.

When BJT is in forward-active operating mode, EBJ is in forward-bias ($V_{BE} > 0$), CBJ is in reverse-bias ($V_{BC} < 0$). I_C, I_B, and I_E of BJT after base-width modulation are given by:



$$I_C = I_s \cdot e^{\frac{V_{BE}}{V_T}} \left(1 + \frac{V_{CE}}{V_A}\right)$$
$$I_B = \frac{I_C}{\beta_f}$$
$$I_E = I_c + I_B \tag{6}$$

By partial differentiation, we can obtain:

$$\frac{\partial I_C}{\partial V_{BE}} = g_m = \frac{I_s}{V_T} \cdot (1 + \frac{V_{CE}}{V_A}) \cdot e^{\frac{V_{BE}}{V_T}}$$
$$\frac{\partial I_C}{\partial V_{CE}} = g_{CE} = \frac{I_s}{V_A} \cdot e^{\frac{V_{BE}}{V_T}}$$
$$\frac{\partial I_B}{\partial V_{BE}} = g_{BE} = \frac{I_s}{\beta_f \cdot V_T} \cdot \left(1 + \frac{V_{CE}}{V_A}\right) \cdot e^{\frac{V_{BE}}{V_T}} \tag{7}$$

Combining Equation set 7 and Equation set 3 gives an equivalent model for BJT in forward-active mode:

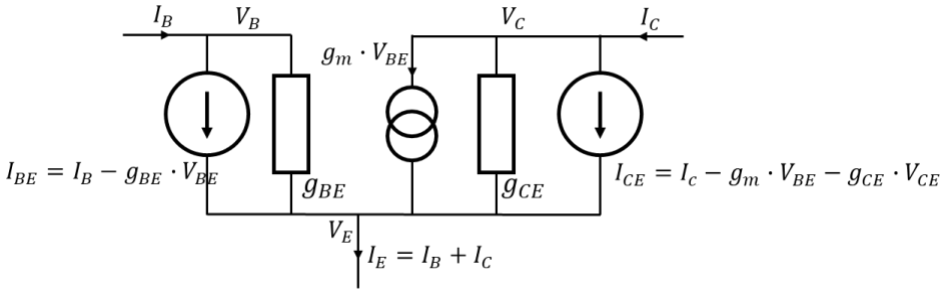
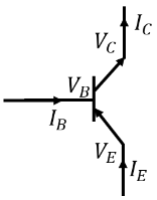


Figure 5: Equivalent model for BJT in forward-active mode

When BJT is in reverse-active mode, EBJ is in reverse-bias ($V_{BE} < 0$) and CBJ is in forward-bias ($V_{BC} > 0$). Currents related to BJT are given by [8]:



$$I_E = I_s \cdot e^{\frac{V_{BC}}{V_T}}$$
$$I_B = \frac{I_s}{\beta_r} \cdot \left(e^{\frac{V_{BC}}{V_T}} - 1 \right)$$
$$I_C = I_B + I_E \tag{8}$$

Since EBJ and CBJ of a BJT are not symmetrical, there is a difference between current gain in forward-active mode β_f and reverse-active mode β_r. In forward-active mode, electrons flowing into base are mostly captured by collector, while in reverse-active mode, only a small amount of electrons flowing into base are received by emitter. Therefore, β_f is usually greater than 100, while β_r is normally between 0.1 to 5.

From partial differentiation,

$$\frac{\partial I_E}{\partial V_{BC}} = g_m = \frac{I_s}{V_T} \cdot e^{\frac{V_{BC}}{V_T}}$$
$$\frac{\partial I_B}{\partial V_{BC}} = g_{BC} = \frac{I_s}{\beta_r \cdot V_T} \cdot e^{\frac{V_{BC}}{V_T}} \tag{9}$$

The equivalent model of BJT in reverse-active mode is:

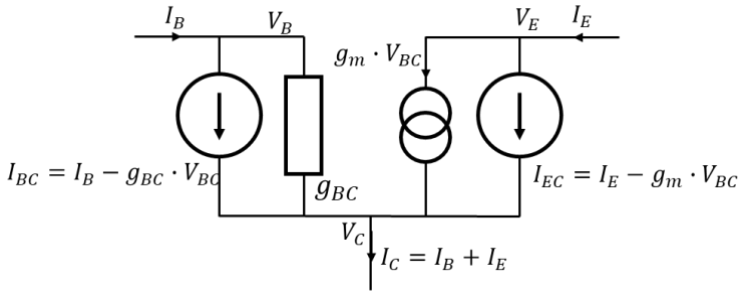
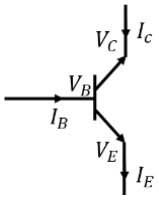


Figure 6: Equivalent model for BJT in reverse-active mode

When BJT is in saturation mode, EBJ is in forward bias ($V_{BE} > 0$) and CBJ is in forward bias ($V_{BC} > 0$). The equations of currents related to BJT is given by [9]:



$$I_c = I_s \cdot e^{\frac{V_{BE}}{V_T}} - \frac{I_s}{\alpha_r} e^{\frac{V_{BC}}{V_T}}$$
$$I_B = \frac{I_s}{\beta_f} \cdot e^{\frac{V_{BE}}{V_T}} + \frac{I_s}{\alpha_r} e^{\frac{V_{BC}}{V_T}}$$
$$I_E = I_B + I_C \tag{10}$$

By partial differentiation,

$$\frac{\partial I_C}{\partial V_{BE}} = g_m = \frac{I_s}{V_T} \cdot e^{\frac{V_{BE}}{V_T}}$$
$$\frac{\partial I_C}{\partial V_{CB}} = g_{CB} = \frac{I_s}{V_T \cdot \alpha_r} \cdot e^{\frac{V_{BC}}{V_T}}$$
$$\frac{\partial I_B}{\partial V_{BE}} = g_i = \frac{I_s}{\beta_f \cdot V_T} \cdot e^{\frac{V_{BE}}{V_T}} \tag{11}$$

The equivalent model of BJT in saturation mode is:

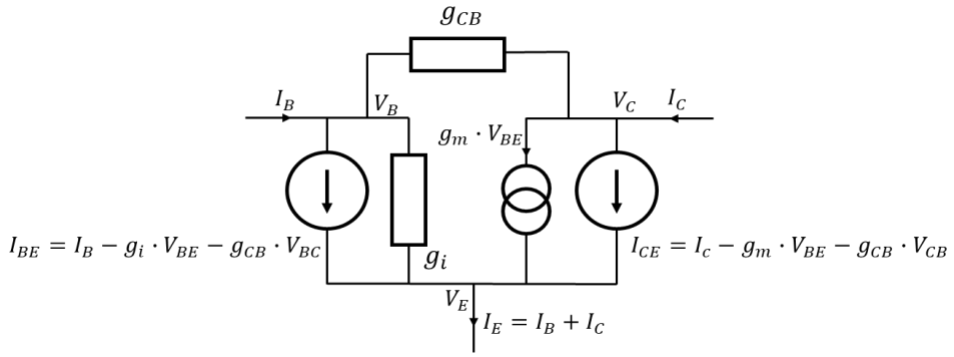


Figure 7: Equivalent model for BJT in saturation mode

When BJT is in cut-off mode, EBJ is in reverse bias ($V_{BE} < 0$) and CBJ is in reverse bias ($V_{BC} < 0$). $I_C \approx I_B \approx I_E \approx 0$, BJT is equivalent to open circuit.

In order to simplify the implementation of the equivalent model conversion, we combine the three models (one for each mode) mentioned above into an integrated model. This model can represent the situations in all modes.

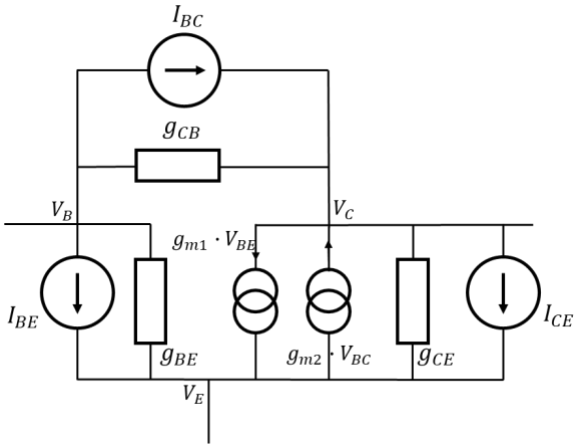


Figure 8: Overall model for BJT

Forward-active mode: $g_{CB} = 0,\ I_{BC} = 0, g_{m2} = 0$

Reverse-active mode: $g_{CE} = 0, g_{BE} = 0, I_{BE} = 0, g_{m1} = 0$

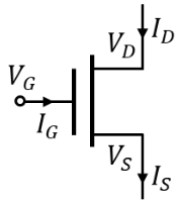
Saturation mode: $g_{CE} = 0, g_{m2} = 0,\ I_{BC} = 0$

Cut-off mode: all components are equal to open circuit.

And the values of other components are calculated using their own equations in different modes.

3.2.3 MOSFET

MOSFETS have 3 operating modes: triode mode, saturation mode, and cut-off mode.



$$I_D = I_S$$
$$I_G = 0$$

When MOSFET is in triode mode [10]:

$$I_D = k \cdot [2(V_{GS} - V_t) \cdot V_{DS} - {V_{DS}}^2]$$

By partial differentiation,

$$\frac{\partial I_D}{\partial V_{GS}} = g_m = 2k \cdot V_{DS}$$

$$\frac{\partial I_D}{\partial V_{DS}} = g_o = 2k \cdot (V_{GS} - V_{DS} - V_t) \quad (12)$$

When MOSFET is in saturation mode [10]:

$$I_D = k \cdot (V_{GS} - V_t)^2 \cdot (1 + \frac{V_{DS}}{V_A})$$

By partial differentiation,

$$\frac{\partial I_D}{\partial V_{GS}} = g_m = 2k \cdot (V_{GS} - V_t)$$

$$\frac{\partial I_D}{\partial V_{DS}} = g_o = \frac{k}{V_A} \cdot (V_{GS} - V_t)^2 \quad (13)$$

From Equation set 12 and Equation set 13, we can obtain the equivalent model of MOSFET. The values of g_m and g_o are dependent on different modes of the MOSFET [11].

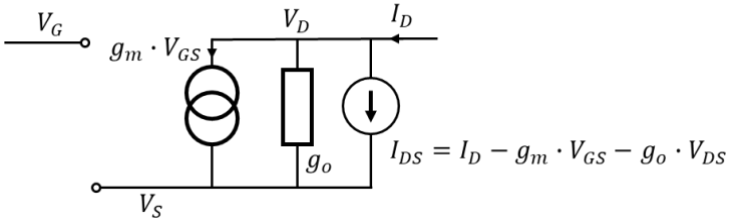


Figure 9: Equivalent model for MOSFET

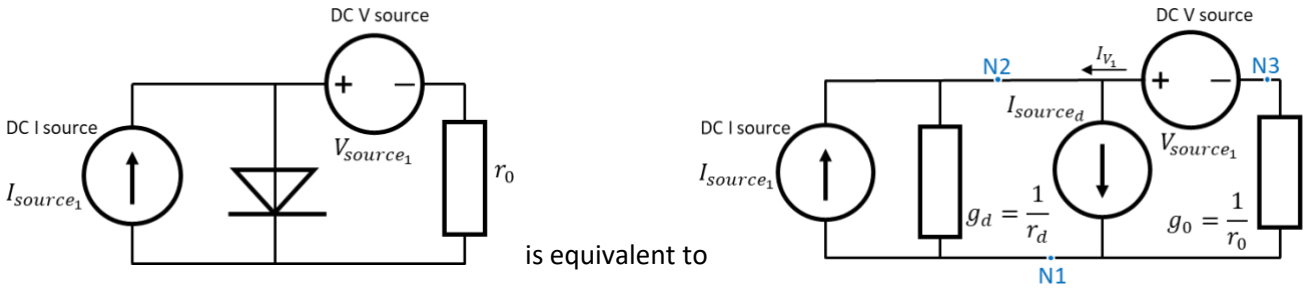
When MOSFET is in cut-off mode, $I_D = I_S \approx 0$. MOSFET is equivalent to open circuit.

After replacing all non-linear devices by their equivalent models, we can form the Jacobian matrix $Jr(\boldsymbol{x}_n)$ by putting the conductance into the correct row-column location. For linear devices, conductance can be used directly, but for non-linear devices, we should use the conductance of its equivalent models (i.e., resistors & voltage-controlled current sources).

RHS column $Jr(\boldsymbol{x}_n) \cdot \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_n)$ is formed by the addition of equivalent current sources in non-linear device models and the constant current sources flowing into each node. This is similar to the MNA method shown in the previous section. Note that if the circuit contains m DC voltage sources, the last m rows and the last m columns of the Jacobian matrix are set in the same way as the submatrix $\boldsymbol{C}$ and $\boldsymbol{B}$, which is shown in Section 3.1 MNA analysis. The last m rows of the RHS column are set by the value of DC voltage sources.

By solving the Jacobian matrix system, we can obtain the next approximation $\boldsymbol{x}_{n+1}$, then we can use $\boldsymbol{x}_{n+1}$ to build the Jacobian system again and obtain the next approximation $\boldsymbol{x}_{n+2}$… The iteration stops until the approximation converges to the desired precision. The final solution will be the DC operating point for the circuit which is useful for AC analysis at a later stage.

Take the following circuit as an example:



Combining Newton-Raphson method and MNA analysis, the matrix system corresponding to the circuit is:

$$\boldsymbol{x}_{n+1} \cdot Jr(\boldsymbol{x}_n) = Jr(\boldsymbol{x}_n) \cdot \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_n)$$

$$Jr(\boldsymbol{x}_n) = \begin{array}{c} \\ \text{N1} \\ \text{N2} \\ \text{N3} \\ \text{C} \end{array} \begin{array}{c} \begin{array}{cccc} \text{G} & \text{N1} & \text{N2} & \text{N3} \end{array} \\ \begin{pmatrix} g_d + g_0 & -g_d & -g_0 & 0 \\ -g_d & g_d & 0 & 1 \\ -g_0 & 0 & g_0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \end{array} \begin{array}{c} \\ \\ \\ \\ \text{B} \end{array}$$

, where $g_d(\boldsymbol{x}_n) = \frac{I_s}{V_T} \cdot e^{\frac{V_{2_n} - V_{1_n}}{v_T}}$.

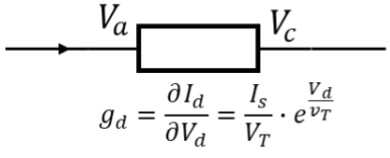
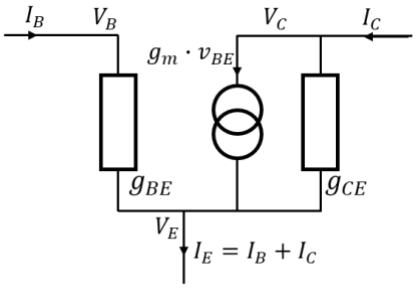
$$Jr(\boldsymbol{x}_n) \cdot \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_n) = \begin{pmatrix} -I_{source_1} + I_{source_d} + I_{V_1} \\ I_{source_1} - I_{source_d} \\ -I_{V_1} \\ V_{source_1} \end{pmatrix}$$

, where $I_{source_d}(\boldsymbol{x}_n) = I_s \cdot \left(e^{\frac{V_{21_n}}{V_T}} - 1 \right) - \frac{I_s}{V_T} \cdot e^{\frac{V_{21_n}}{V_T}} \cdot V_{21_n}$.

Hence, we can find the next iteration $\boldsymbol{x}_{n+1} = Jr(\boldsymbol{x}_n)^{-1} \cdot (Jr(\boldsymbol{x}_n) \cdot \boldsymbol{x}_n - \boldsymbol{F}(\boldsymbol{x}_n))$, where $\boldsymbol{x}_{n+1} = \begin{pmatrix} V_{1_{n+1}} \\ V_{2_{n+1}} \\ V_{3_{n+1}} \\ I_{V_1} \end{pmatrix}$.

3.3 AC Analysis

In AC analysis, DC voltage sources are treated as short circuits, and DC current sources are treated as open circuits. Therefore, the equivalent models for non-linear devices shown in the previous section need some modifications: the equivalent DC current sources in the non-linear device models need to be removed. Therefore, the models of non-linear devices for AC analysis are:

Diode	 $I_d = I_s \cdot \left(e^{\frac{V_d}{V_T}} - 1\right)$ $\Delta I_d = \frac{\partial I_d}{\partial V_d} \cdot \Delta V_d$ $g_d = \frac{\partial I_d}{\partial V_d} = \frac{I_s}{V_T} \cdot e^{\frac{V_d}{V_T}}$
BJT in forward-active mode	 $\Delta I_C = \frac{\partial I_C}{\partial V_{BE}} \cdot \Delta V_{BE} + \frac{\partial I_C}{\partial V_{CE}} \cdot \Delta V_{CE}$ $\Delta I_B \approx \frac{\partial I_B}{\partial V_{BE}} \cdot \Delta V_{BE}$ $g_m = \frac{\partial I_C}{\partial V_{BE}} = \frac{I_s}{V_T} \cdot (1 + \frac{V_{CE}}{V_A}) \cdot e^{\frac{V_{BE}}{V_T}}$ $g_{CE} = \frac{\partial I_C}{\partial V_{CE}} = \frac{I_s}{V_A} \cdot e^{\frac{V_{BE}}{V_T}}$ $g_{BE} = \frac{\partial I_B}{\partial V_{BE}} = \frac{I_s}{\beta_f \cdot V_T} \cdot (1 + \frac{V_{CE}}{V_A}) \cdot e^{\frac{V_{BE}}{V_T}}$

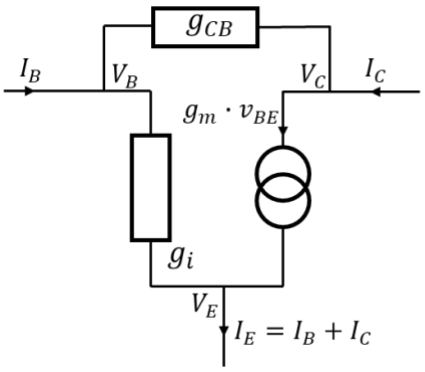
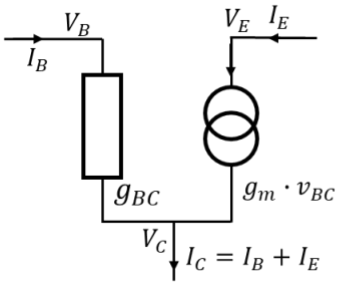
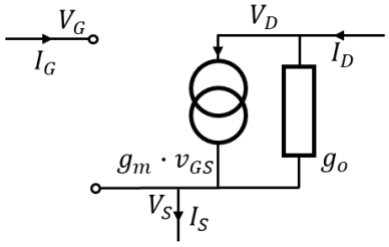
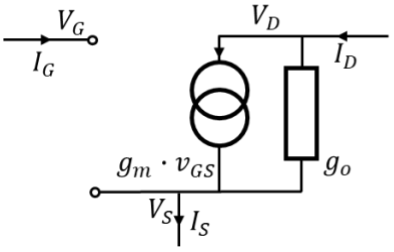
BJT in saturation mode	 $\Delta I_C = \frac{\partial I_C}{\partial V_{BE}} \cdot \Delta V_{BE} + \frac{\partial I_C}{\partial V_{CB}} \cdot \Delta V_{CB}$ $\Delta I_B \approx \frac{\partial I_B}{\partial V_{BE}} \cdot \Delta V_{BE} + \frac{\partial I_B}{\partial V_{BC}} \cdot \Delta V_{BC}$ $g_m = \frac{\partial I_C}{\partial V_{BE}} = \frac{I_s}{V_T} \cdot e^{\frac{V_{BE}}{V_T}}$ $g_{CB} = \frac{\partial I_C}{\partial V_{CB}} = \frac{I_s}{V_T \cdot \alpha_r} \cdot e^{\frac{V_{BC}}{V_T}}$ $g_{BE} = \frac{\partial I_B}{\partial V_{BE}} = \frac{I_s}{\beta_f \cdot V_T} \cdot e^{\frac{V_{BE}}{V_T}}$
BJT in reverse-active mode	 $\Delta I_E = \frac{\partial I_E}{\partial V_{BC}} \cdot \Delta V_{BC}$ $\Delta I_B = \frac{\partial I_B}{\partial V_{BC}} \cdot \Delta V_{BC}$ $g_m = \frac{\partial I_E}{\partial V_{BC}} = \frac{I_s}{V_T} \cdot e^{\frac{V_{BC}}{V_T}}$ $g_{BC} = \frac{\partial I_B}{\partial V_{BC}} = \frac{I_s}{\beta_r \cdot V_T} \cdot e^{\frac{V_{BC}}{V_T}}$
MOSFET in saturation mode	 $\Delta I_D = \frac{\partial I_D}{\partial V_{GS}} \cdot \Delta V_{GS} + \frac{\partial I_D}{\partial V_{DS}} \cdot \Delta V_{DS}$ $g_m = \frac{\partial I_D}{\partial V_{GS}} = 2k \cdot (V_{GS} - V_t)$ $g_o = \frac{\partial I_D}{\partial V_{DS}} = \frac{k}{V_A} \cdot (V_{GS} - V_t)^2$
MOSFET in triode mode	 $\Delta I_D = \frac{\partial I_D}{\partial V_{GS}} \cdot \Delta V_{GS} + \frac{\partial I_D}{\partial V_{DS}} \cdot \Delta V_{DS}$ $g_m = \frac{\partial I_D}{\partial V_{GS}} = 2k \cdot V_{DS}$ $g_o = \frac{\partial I_D}{\partial V_{DS}} = 2k \cdot (V_{GS} - V_{DS} - V_t)$

Table 2: Equivalent models of non-linear devices in AC analysis

By replacing non-linear devices with their equivalent models and using the conductance of components to build a conductance matrix, AC analysis can be done using the MNA method explained in Section 3.1.

4 Implementation and Testing

4.1 Classes

4.1.1 Overall Class Diagram

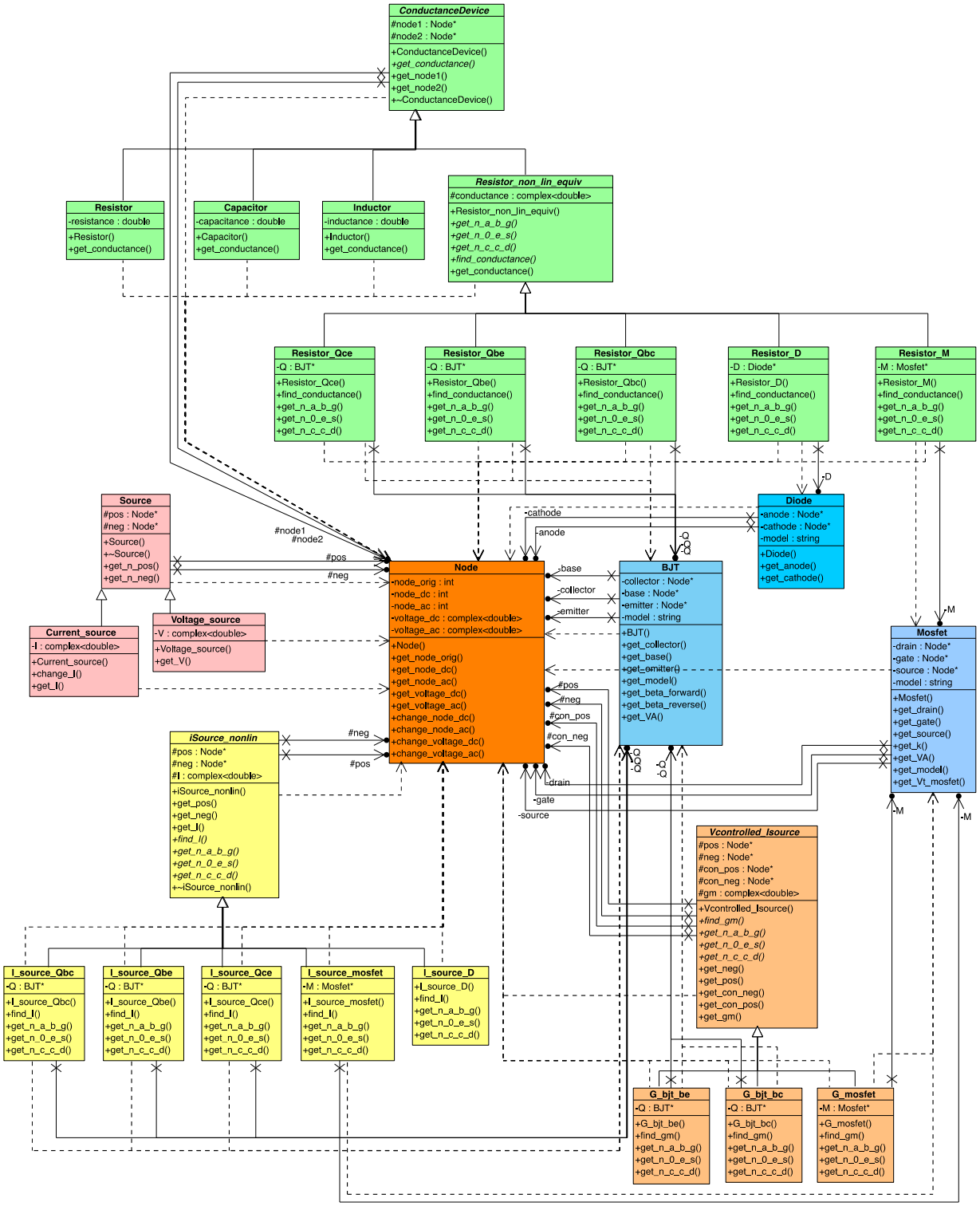
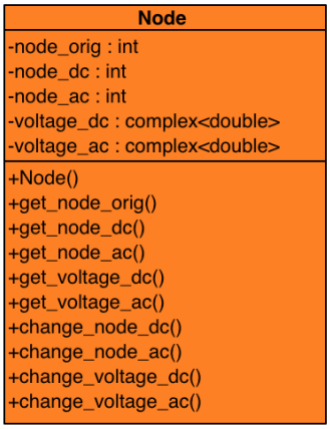


Figure 10: Class diagram

4.1.2 Class Node

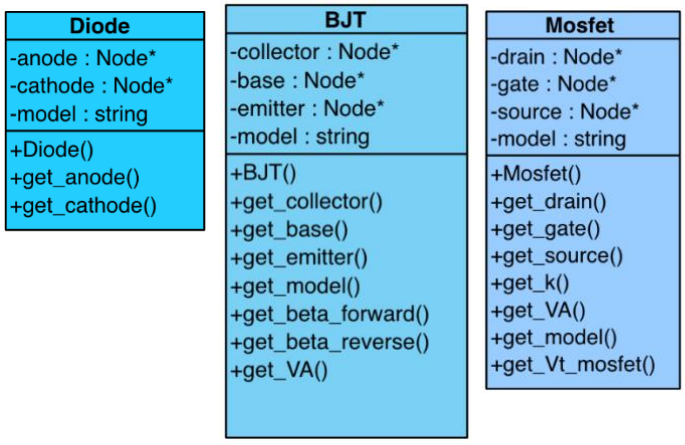


Function names in the form '**get_ATTRIBUTE**' will return the specified attribute in its originally defined type. Function names in the form '**change_ATTRIBUTE**' will change the value of the specified attribute.

Attributes:

- **node_orig**: stores the original node number. Read only and won't be changed throughout the program
- **node_dc**: initially the same as **node_orig**. It is changed when short-circuiting inductors and AC voltage sources for DC analysis.
- **node_ac**: initially the same as **node_orig**. It is changed when short-circuiting DC voltage sources for AC analysis.
- **voltage_dc**: stores the voltage calculated from DC analysis
- **voltage_ac**: stores the voltage calculated from AC analysis

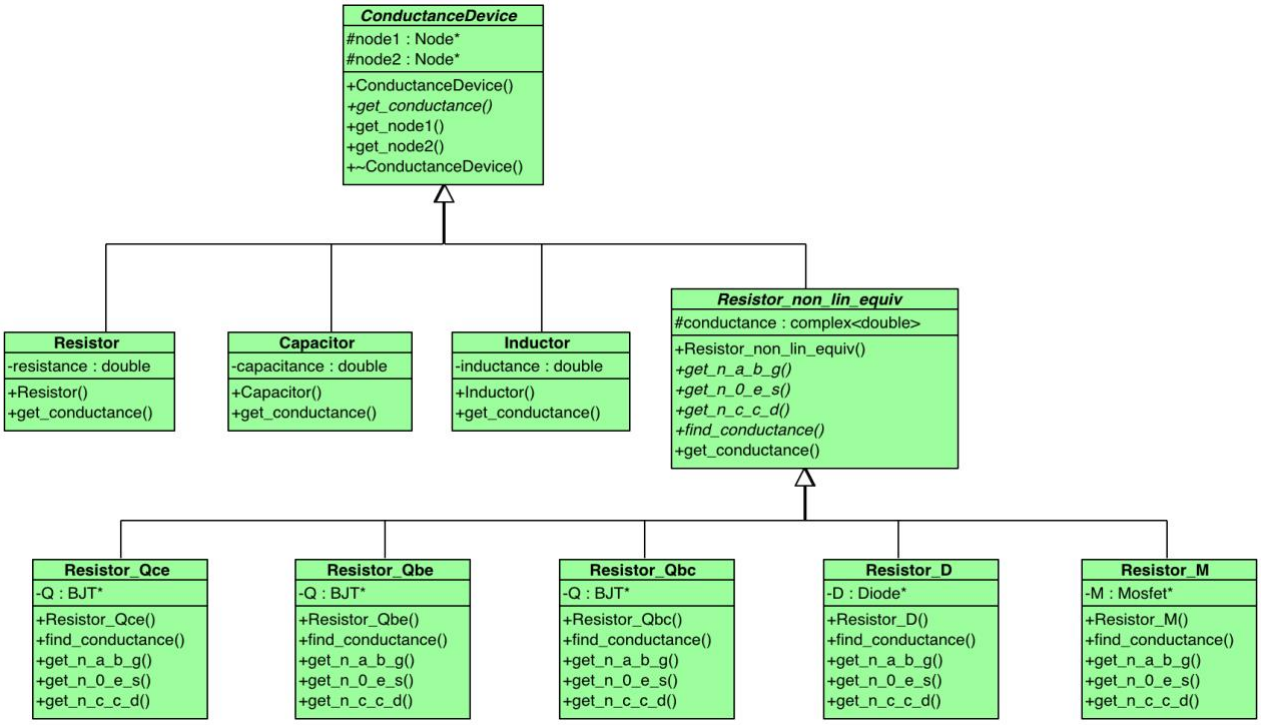
4.1.3 Class Diode/BJT/MOSFET



These are 3 separate classes which does not belong to any class families (no parent class nor child class).

Since BJT and MOSFET have different types, (i.e., NPN/PNP, NMOS/PMOS), and the parameters related to different types are different, an attribute called **model** is used to store the type of the non-linear device. This allows us to use **model** as a condition, to make **get_CONSTANT** return different values for different models accordingly. For information about the constants, see Section 2.1.4.

4.1.4 Class Conductance Device



Resistor, **Capacitor**, **Inductor** all have the function '**std::complex<double> get_conductance(double omega)**'. This function uses the corresponding formula and the specified ω to calculate the conductance.

- Resister: $conductance = \frac{1}{R}$
- Capacitor: $conductance = (\,0\,, \omega C\,)$
- Inductor: $conductance = \left(\,0\,, -\frac{1}{\omega L}\right)$

Resistor_Qce, **Resistor_Qbe**, **Resistor_Qbc**, **Resistor_D**, **Resistor_M** all have the function **void find_conductance(std::complex<double> v_a_b_g, std::complex<double> v_0_e_s, std::complex<double> v_c_c_d)**.

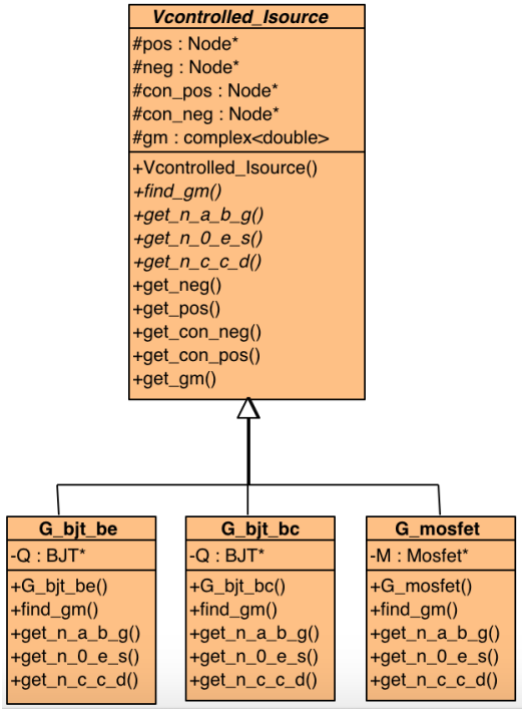
- **v_a_b_g** represents: V_{anode} (Diode), V_{base} (BJT), V_{gate}(MOSFET)
- **v_0_e_s** represents: unused (Diode), $V_{emitter}$ (BJT), V_{source} (MOSFET)
- **v_c_c_d** represents: $V_{cathode}$ (Diode), $V_{collector}$ (BJT), V_{drain} (MOSFET)

This function (**find_conductance**) uses the corresponding formula with the given voltages to calculate the conductance of each non-linear equivalent resistor. And it changes the value of the **conductance** attribute so that the new conductance can be retrieved using **get_conductance()** later on.

- **Resistor_D**:
  - On: $g_d = \frac{I_S}{V_T} \cdot e^{\frac{V_d}{vT}}$ from Equation set 5
  - Off: 0
- **Resistor_Qce**:
  - Active: $\frac{\partial I_C}{\partial V_{CE}} = g_o = \frac{I_S}{V_A} \cdot e^{\frac{V_{BE}}{V_T}}$ from Equation set 7
  - Saturation: 0
  - Cut-off: 0
  - Reverse: 0

- **Resistor_Qbe**:
  - Active: $\frac{\partial I_B}{\partial V_{BE}} = g_i = \frac{I_s}{\beta_f \cdot V_T} \cdot \left(1 + \frac{V_{CE}}{V_A}\right) \cdot e^{\frac{V_{BE}}{V_T}}$ from Equation set 7
  - Saturation: $\frac{\partial I_B}{\partial V_{BE}} = g_i = \frac{I_s}{\beta_f \cdot V_T} \cdot e^{\frac{V_{BE}}{V_T}}$ from Equation set 11
  - Cut-off: 0
  - Reverse: 0
- **Resistor_Qbc**:
  - Active: 0
  - Saturation: $\frac{\partial I_C}{\partial V_{CB}} = g_{CB} = \frac{I_s}{V_T \cdot \alpha_r} \cdot e^{\frac{V_{BC}}{V_T}}$ from Equation set 11
  - Cut-off: 0
  - Reverse: $\frac{\partial I_B}{\partial V_{BC}} = g_i = \frac{I_s}{\beta_r \cdot V_T} \cdot e^{\frac{V_{BC}}{V_T}}$ from Equation set 9
- **Resistor_M**:
  - Linear: $\frac{\partial I_D}{\partial V_{DS}} = g_o = 2k \cdot (V_{GS} - V_{DS} - V_t)$ from Equation set 12
  - Saturation: $\frac{\partial I_D}{\partial V_{DS}} = g_o = \frac{k}{V_A} \cdot (V_{GS} - V_t)^2$ from Equation set 13
  - Cut-off: 0

4.1.5 Class Vcontrolled_Isource (Voltage Controlled Current Source)



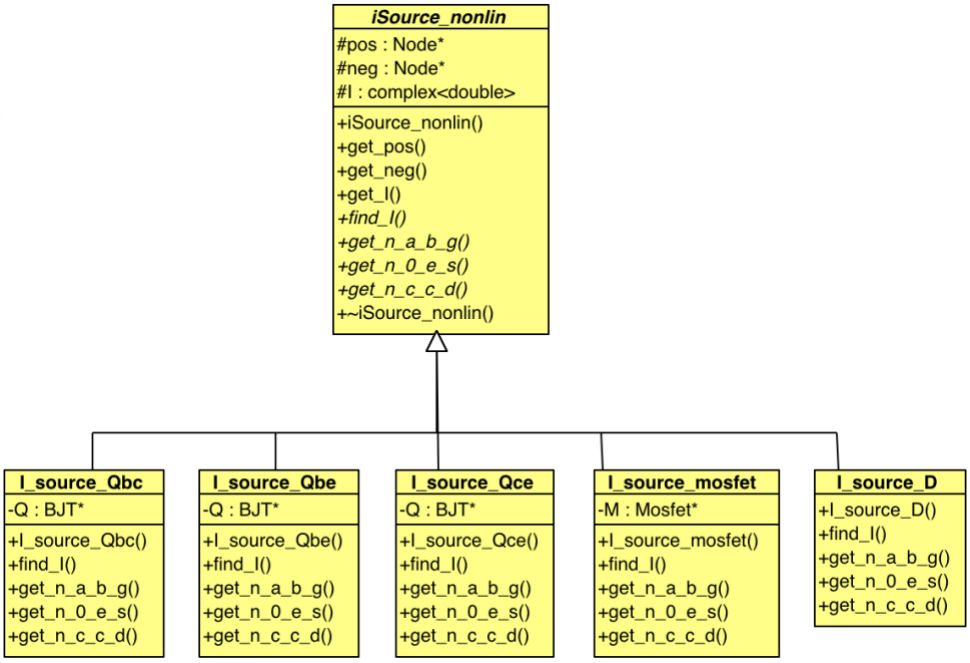
Attributes **pos** and **neg** hold the nodes of its 2 terminals. Attributes **con-pos** and **con_neg** hold the 2 nodes that control the current source.

G_bjt_be, **G_bjt_bc**, **G_mosfet** all have the function **void find_gm(std::complex<double> v_a_b_g, std::complex<double> v_0_e_s, std::complex<double> v_c_c_d)**.

This function uses the corresponding formula with the given voltages to calculate and then store gm for each voltage controlled current source in non-linear equivalent models.

- **G_bjt_be**:
  - Active: $\frac{\partial I_C}{\partial V_{BE}} = g_m = \frac{I_S}{V_T} \cdot (1 + \frac{V_{CE}}{V_A}) \cdot e^{\frac{V_{BE}}{V_T}}$ from Equation set 7
  - Saturation: $\frac{\partial I_C}{\partial V_{BE}} = g_m = \frac{I_S}{V_T} \cdot e^{\frac{V_{BE}}{V_T}}$ from Equation set 11
  - Cut-off: 0
  - Reverse: 0
- **G_bjt_bc**:
  - Active: 0
  - Saturation: 0
  - Cut-off: 0
  - Reverse: $\frac{\partial I_E}{\partial V_{BC}} = g_m = \frac{I_S}{V_T} \cdot e^{\frac{V_{BC}}{V_T}}$ from Equation set 9
- **G_mosfet**:
  - Active: $\frac{\partial I_D}{\partial V_{GS}} = g_m = 2k \cdot V_{DS}$ from Equation set 12
  - Saturation: $\frac{\partial I_D}{\partial V_{GS}} = g_m = 2k \cdot (V_{GS} - V_t)$ from Equation set 13
  - Cut-off: 0

4.1.6 Class iSource_nonlin (Non-linear Equivalent Current Source)

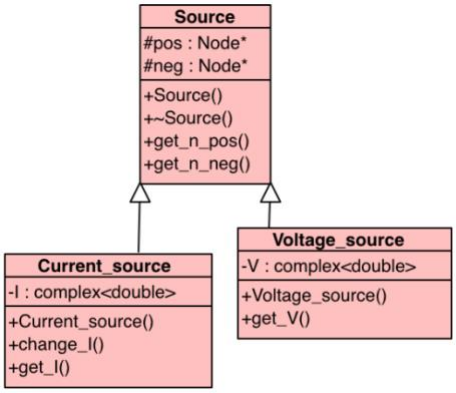


The child classes all have the function **void find_I(std::complex<double> v_a_b_g, std::complex<double> v_0_e_s, std::complex<double> v_c_c_d)**. This function uses the corresponding formula with the given voltages to calculate and then store the value of current for each non-linear equivalent current source.

- **I_source_D**:
  - On: $I_{source} = I_s \cdot \left(e^{\frac{V_d}{V_T}} - 1 \right) - \frac{I_s}{V_T} \cdot e^{\frac{V_d}{V_T}} \cdot V_d$ from Figure 3
  - Off: 0

- **I_source_mosfet**:
  - Linear: $I_{DS} = I_D - g_m V_{GS} - g_o V_{DS}$
    $$= k \cdot \left[2(V_{GS} - V_t) \cdot V_{DS} - {V_{DS}}^2\right] - 2k \cdot V_{DS} \cdot V_{GS} - 2k \cdot (V_{GS} - V_{DS} - V_t) \cdot V_{DS}$$
    from Figure 7 & Equation set 12
  - Saturation: $I_{DS} = I_D - g_m V_{GS} - g_o V_{DS}$
    $$= k \cdot (V_{GS} - V_t)^2 \cdot (1 + \frac{V_{DS}}{V_A}) - 2k \cdot (V_{GS} - V_t) \cdot V_{GS} - \frac{k}{V_A} \cdot (V_{GS} - V_t)^2 \cdot V_{DS}$$
    from Figure 7 & Equation set 13
  - Cut-off mode: 0
- **I_source_Qce**:
  - Active: $I_{CE} = I_C - g_m V_{BE} - g_o V_{CE}$
    $$= I_s \cdot e^{\frac{V_{BE}}{V_T}} \left(1 + \frac{V_{CE}}{V_A}\right) - \frac{I_s}{V_T} \cdot \left(1 + \frac{V_{CE}}{V_A}\right) \cdot e^{\frac{V_{BE}}{V_T}} \cdot V_{BE} - \frac{I_s}{V_A} \cdot e^{\frac{V_{BE}}{V_T}} \cdot V_{CE}$$
    from Figure 4 & Equation set 6 & 7
  - Saturation: $I_{CE} = I_C - g_m V_{BE} - g_{CB} V_{CB}$
    $$= I_s \cdot e^{\frac{V_{BE}}{V_T}} - \frac{I_s}{\alpha_r} e^{\frac{V_{BC}}{V_T}} - \frac{I_s}{V_T} \cdot e^{\frac{V_{BE}}{V_T}} \cdot V_{BE} - \frac{I_s}{V_T \cdot \alpha_r} \cdot e^{\frac{V_{BC}}{V_T}} \cdot V_{CB}$$
    from Figure 6 & Equation set 10 & 11
  - Reverse: $I_{CE} = g_m V_{BC} - I_E$
    $$= \frac{I_s}{V_T} \cdot e^{\frac{V_{BC}}{V_T}} \cdot V_{BC} - I_s \cdot e^{\frac{V_{BC}}{V_T}}$$
    from Figure 5 & Equation set 8 & 9
  - Cut-off: 0
- **I_source_Qbe**:
  - Active: $I_{BE} = I_B - g_i V_{BE}$
    $$= I_s \cdot e^{\frac{V_{BE}}{V_T}} \left(1 + \frac{V_{CE}}{V_A}\right) - \frac{I_s}{V_T} \cdot \left(1 + \frac{V_{CE}}{V_A}\right) \cdot e^{\frac{V_{BE}}{V_T}} \cdot V_{BE} - \frac{I_s}{V_A} \cdot e^{\frac{V_{BE}}{V_T}} \cdot V_{CE}$$
    from Figure 4 & Equation set 6 & 7
  - Saturation: $I_{CE} = I_C - g_m V_{BE} - g_{CB} V_{CB}$
    $$= I_s \cdot e^{\frac{V_{BE}}{V_T}} - \frac{I_s}{\alpha_r} e^{\frac{V_{BC}}{V_T}} - \frac{I_s}{V_T} \cdot e^{\frac{V_{BE}}{V_T}} \cdot V_{BE} - \frac{I_s}{V_T \cdot \alpha_r} \cdot e^{\frac{V_{BC}}{V_T}} \cdot V_{CB}$$
    from Figure 6 & Equation set 10 & 11
  - Reverse: 0
  - Cut-off: 0
- **I_source_Qbc**:
  - Active: 0
  - Saturation: 0
  - Reverse: $I_{BC} = I_B - g_i V_{BC}$
    $$= \frac{I_s}{\beta_r} \cdot \left(e^{\frac{V_{BC}}{V_T}} - 1\right) - \frac{I_s}{\beta_r \cdot V_T} \cdot e^{\frac{V_{BC}}{V_T}} \cdot V_{BC}$$
    From Figure 5 & Equation set 8 & 9
  - Cut-off: 0

4.1.7 Class Source



The value of I and V is in the type of complex number. For DC sources it would only have a real component. For AC sources, we need to transform the voltage from polar form (magnitude & phase) into complex form.

[8]

4.2 Identify operating modes

4.2.1 Design

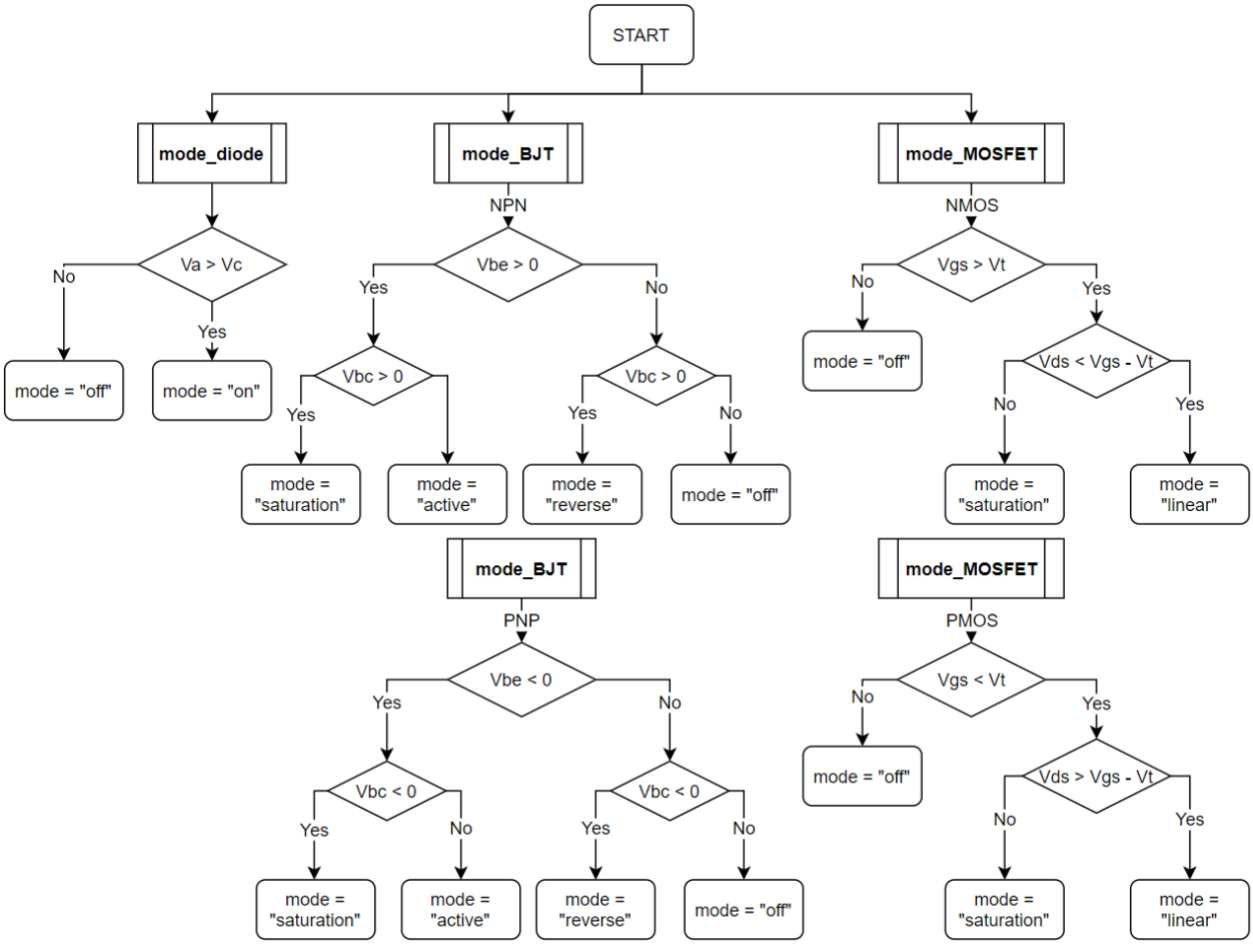


Figure 11: Flowchart for identifying modes of non-linear devices

4.2.2 Testing

Function	Expectation	Problems & Solutions	Examples
Identify the operating modes of non-linear components	Correct modes are printed	Problem: Using abs function to change the complex voltage to double which ignored plus and minus signs. Solution: using real instead of abs.	For BJT, if input $V_C = 1$, $V_B = 0.5$, $V_E = 0$, output would be "active".

4.3 Read File

4.3.1 Design

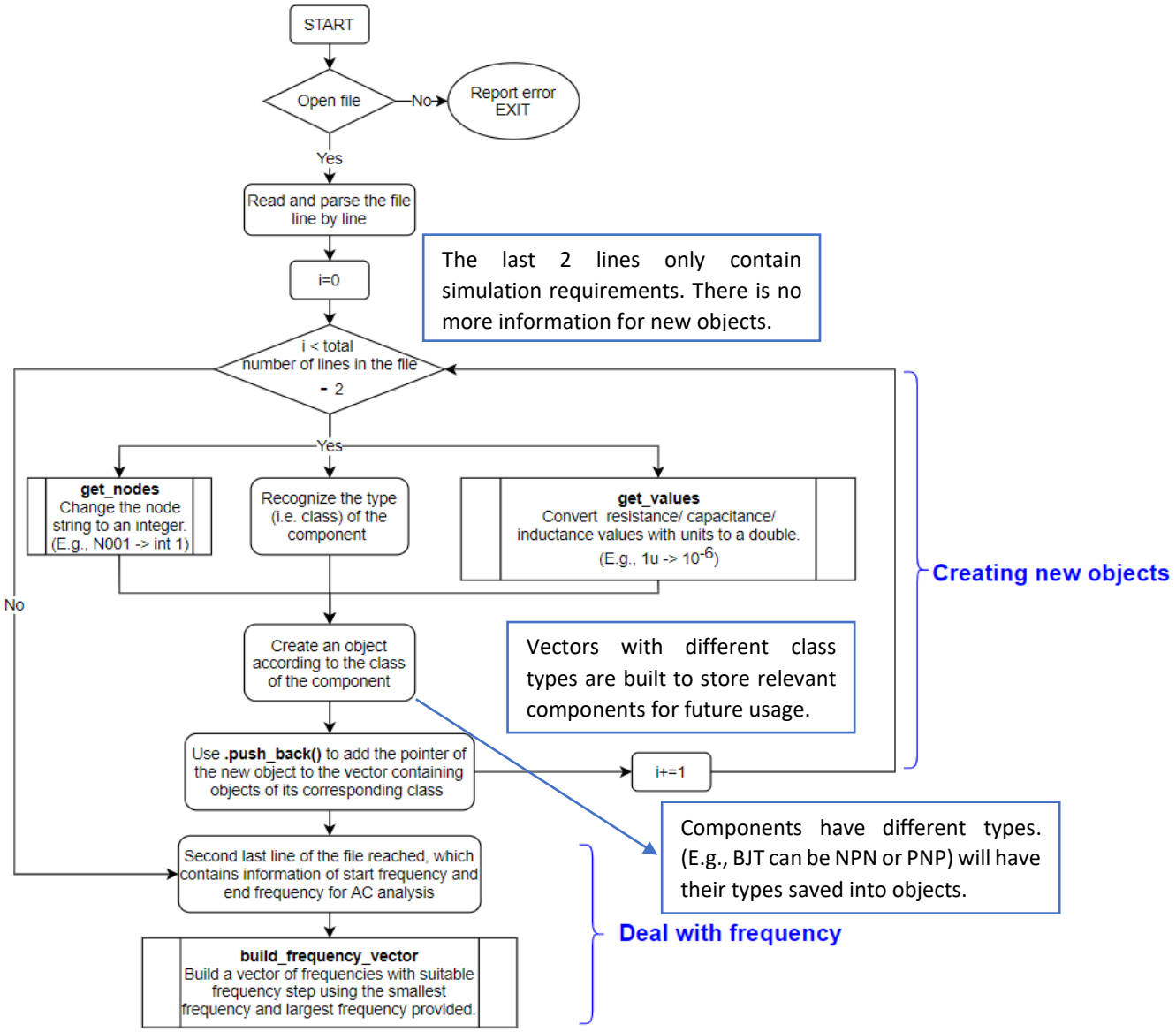


Figure 12: Flowchart for reading netlist

A vector of frequencies with suitable frequency step is built for future use in AC analysis. For $f < 1000Hz$, $\Delta f = 10Hz$. For $f > 1000Hz$, $\Delta f = 1000Hz$. The frequency step is chosen so that there will be enough data points for AC analysis to obtain a smooth curve, while the program will not crush due to overwhelming running data.

4.3.2 Testing

Function	Expectation	Problems & Solutions	Examples
Read file by line	Each line of the file is stored as a string.	N/A	V1 N001 0 5 V2 N004 0 AC 1 0
get_nodes	N is removed, correct integer nodes for each component are printed.	Problem: Initially the conversion is done by extracting the last digit of the node and convert it to an integer. However, this does not work for 2 digits nodes. Solution: Erase the first digit of the node string ("N") instead of only keeping the last digit.	N005 to 5 N010 to 10
get_values	All values of components are printed	N/A	1k to 1000 1u to 1000000

4.4 Short circuit

4.4.1 Design

Short circuit voltage sources:

- AC analysis: short circuit DC voltage sources
- DC analysis: short circuit AC voltage sources

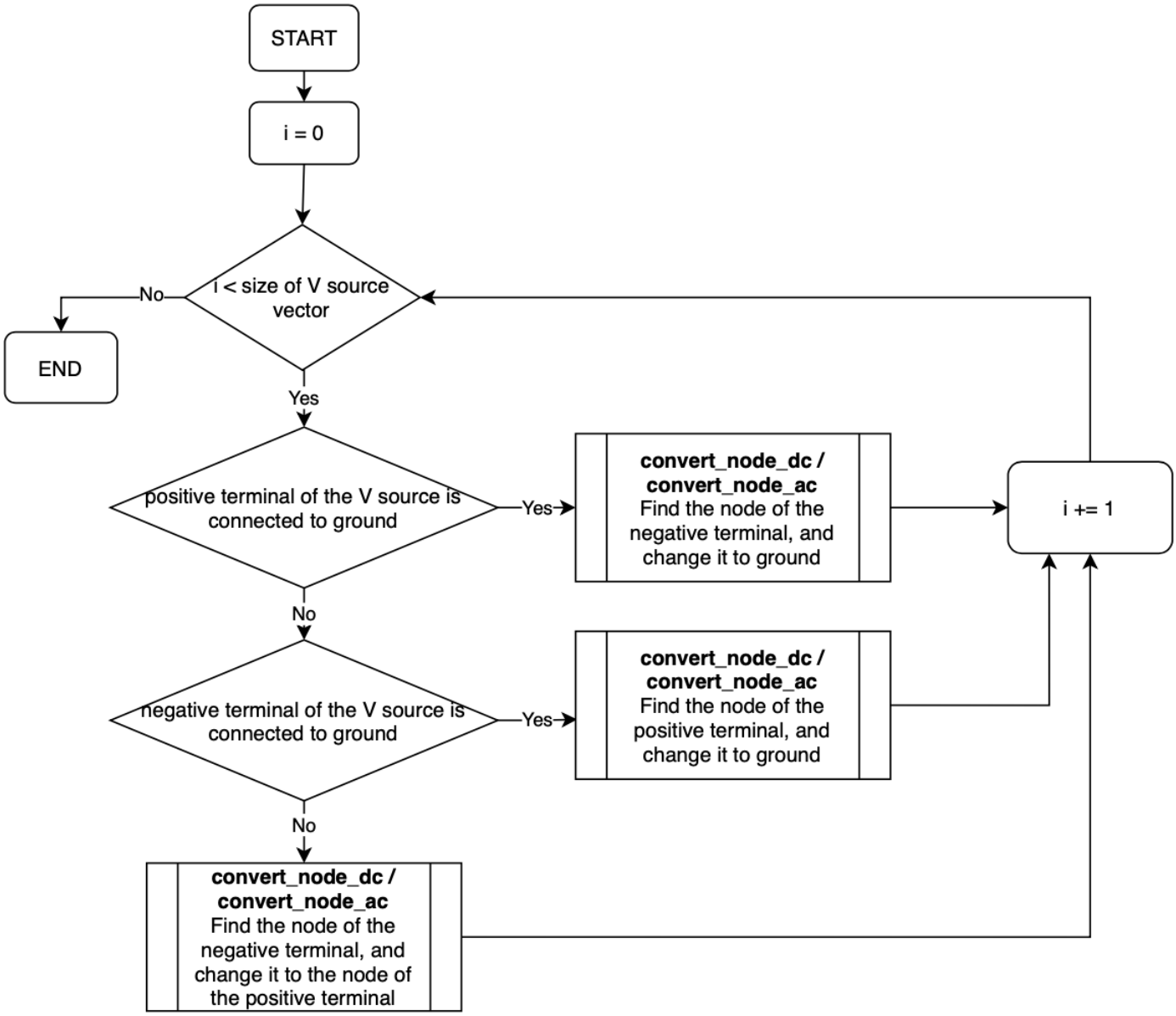


Figure 13: Flowchart for short circuit voltage sources

Short circuit inductors: In DC analysis, we need to short circuit inductors, as when $\omega = 0$, $Z_L = j\omega L = 0$.

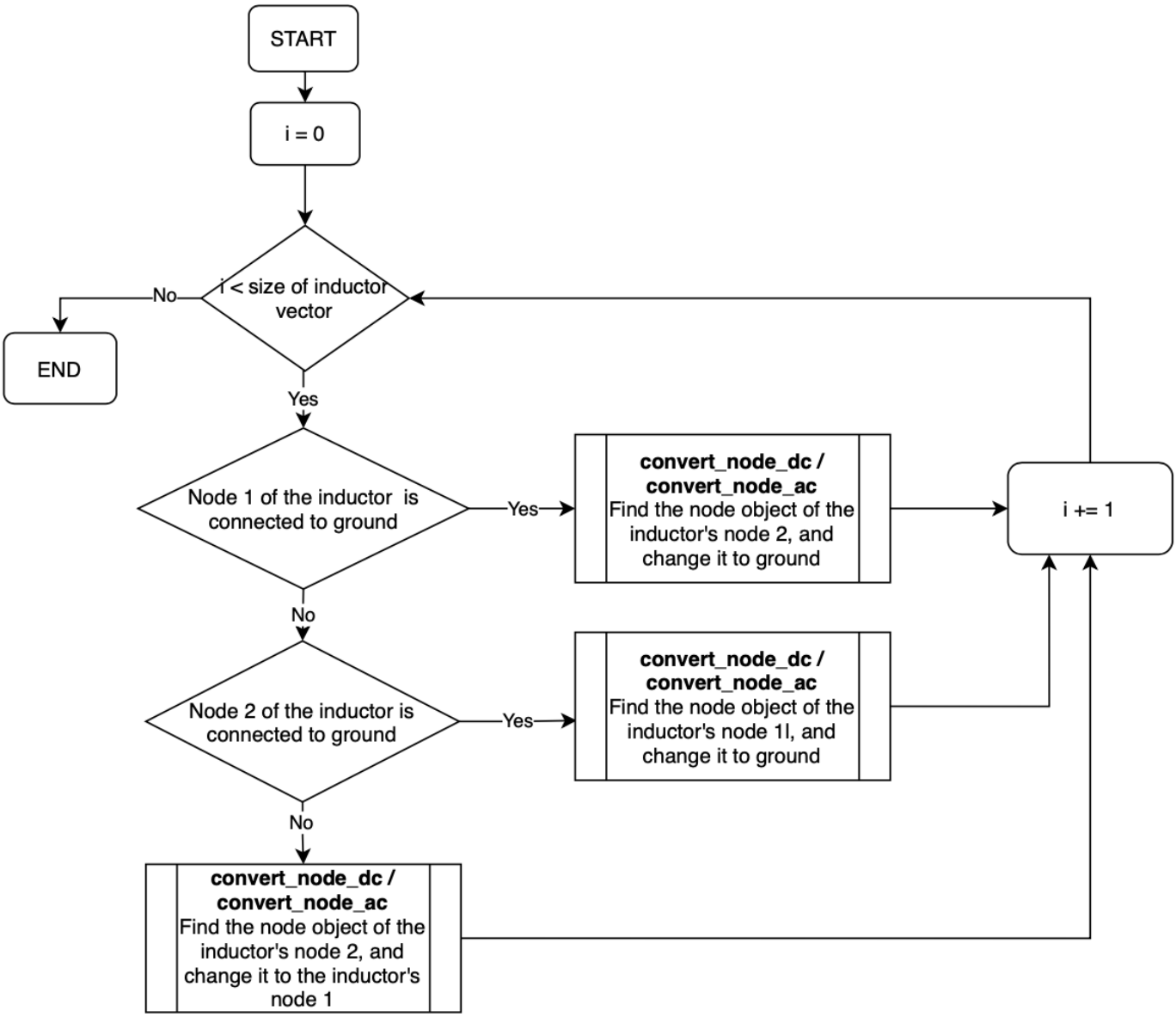


Figure 14: Flowchart for short circuit inductors

These are 2 similar functions that loop through all **Node** object to find node connected to the specified old node, and change it to the new node.

- **void convert_node_ac(std::vector<Node*>& nodes, int old_node, int new_node)**
- **void convert_node_dc(std::vector<Node*>& nodes, int old_node, int new_node)**

4.4.2 Testing

Function	Expectation	Problems & Solutions	Examples
convert_node_dc **convert_node_ac**	The attribute **node_dc/ node_ac** of the node object is changed to the given new value	N/A	**nodes =** [node(2), node(5), node(3), node(1), node(4)] **convert_node_dc(nodes,3,5)** Result: the **node_dc** attribute of node(3) is changed from 3 to 5
			nodes = [node(2), node(4), node(3), node(1)] **convert_node_ac**(**nodes**,2,0) Result: the **node_ac** attribute of node(2) is changed from 2 to 0. Node 0 represents ground.
short_v_source_dc **short_v_source_ac**	For each voltage source in the vector, the attribute **node_dc** of its 2 terminals will become the same	Problem: When a voltage source is connected to ground, the 2 terminal nodes don't always become ground Solution: add if statements to find if the V source is grounded, and make the 2 terminal nodes grounded)	**voltages =** [v(3,5), v(6,1), v(2,0)] **nodes =** [node(3), node(2), node(5), node(1), node(6)] **short_v_source_dc**(**voltages**, **nodes**) Result: node(5) -> node_dc = 3 node(1) -> node_dc = 6 node(2) -> node_dc = 0
short_inductor	For each inductor in the vector, the attribute **node_dc** of its 2 terminals will become the same	Problem: When an inductor is connected to ground, the 2 terminal nodes don't always become ground Solution: add if statements to find if the inductor is grounded, and make the 2 terminal nodes grounded)	**inductors =** [L(5,0), L(2,3)] **nodes =** [node(3), node(2), node(5), node(1)] **short_inductor(inductors, nodes)** Result: node(5) -> node_ac = 0 node(3) -> node_ac = 2

4.5 Open Circuit

- AC analysis: open DC current sources
- DC analysis: open AC current sources and capacitors

Normally, we can simply deal with the open circuit of capacitors by not adding their conductance into the Jacobian matrix. Similarly, we can deal with the open circuit of current sources by not adding their values into the RHS column.

However, there is a special case that we should take note of.

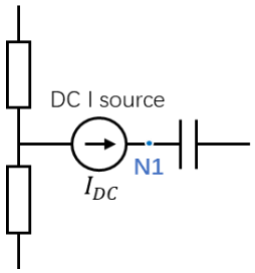


Figure 15: Special case in DC analysis

In DC analysis, if a capacitor is in series with a DC current source, since capacitor is equivalent to open circuit, the N1 KCL equation in the Jacobian matrix system will be: $0V_1 - 0V_2 - 0V_3 - \cdots - 0V_n = I_{DC}$. The equation has no solution and will cause an error to our program. Therefore, we need to change the value of I_{DC} to $0A$ in this case.

4.5.1 Design

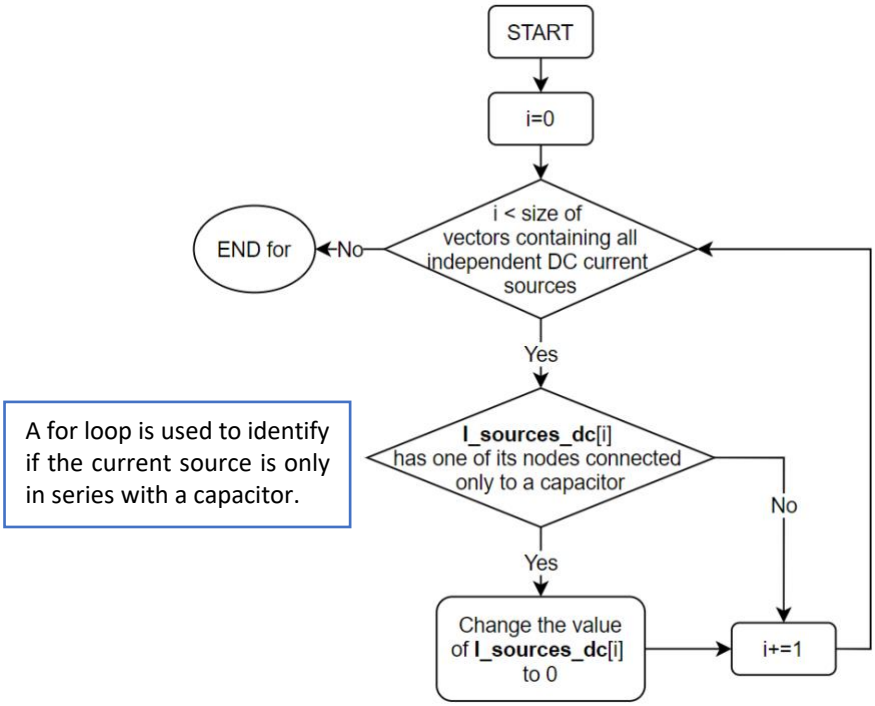


Figure 16: Flowchart for open circuit capacitors

4.5.2 Testing

Function	Expectation	Problems & Solutions	Examples
Open circuit	The values of relevant current sources become 0.	N/A	Current source with 1A of current connected to N2 & N4 Capacitor connected to node N4 & N5 Result: the value of the current sources become 0

4.6 Conversion of Non-Linear Devices to Equivalent Models

4.6.1 Design

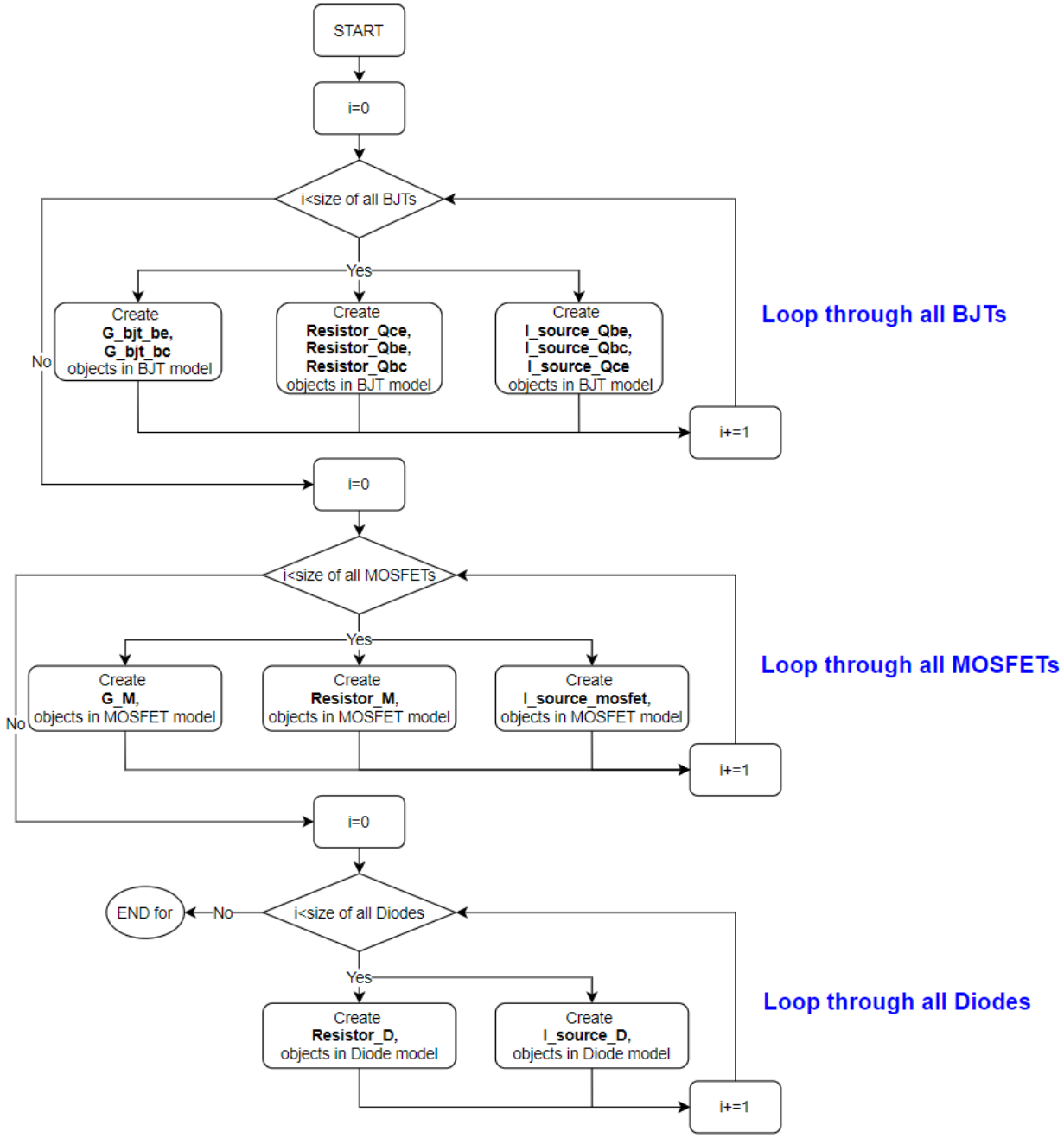


Figure 17: Flowchart for conversion of non-linear devices to equivalent models

4.7 Building Vectors

After converting non-linear devices into their equivalent models, we build vectors to store pointers towards newly-built objects together with originally built objects, so that they can be accessed easily at a later stage when building conductance matrix for DC and AC analysis.

Vector Name	Class Type Contained by Vector	Purpose of the Vector
large_signal_equiv_resistors	**Resistor***, **Resistor_Qce***, **Resistor_Qbc***, **Resistor_Qbe***	Add the conductance of these resistors to the Jacobian matrix in DC analysis.
ssem_equiv_resistors	**Resistor***, **Inductor***, **Capacitor***, **Resistor_Qce***, **Resistor_Qbc***, **Resistor_Qbe***	Add the conductance of these components to the conductance matrix in AC analysis.
non_lin_equiv_resistors	**Resistor_Qce***, **Resistor_Qbc***, **Resistor_Qbe***	Use the **find_conductance** function (operation defined in these classes) to calculate the new conductance of these objects, using the previous approximation of voltage at each node. This is used at the start of each iteration in the Newton-Raphson DC analysis; and it's also used at the start of AC analysis after DC operating point is obtained.
V_controlled_I_sources	**G_bjt_be***, **G_bjt_ce***, **G_M***	Use **find_gm** function inbuilt in these classes to calculate new transconductance during Newton-Raphson iteration and at the start of AC analysis. Need to add g_m into Jacobian matrix/ conductance matrix in DC/ AC analysis.
iSource_nonlin	**I_source_Qbe***, **I_source_Qce***, **I_source_Qbc***	Use **find_I** function inbuilt in these classes to calculate new current value of these equivalent current sources during Newton-Raphson iteration in DC analysis.
iSource_dc	**I_source_Qbe***, **I_source_Qce***, **I_source_Qbc***, **Current_source*** (DC)	Add the value of current sources into RHS column during DC analysis.
iSource_ac	**Current_source*** (AC)	Add the value of AC current sources into RHS column during AC analysis.
vSource_dc	**Voltage_source*** (DC)	Add the value of DC voltage sources into RHS column during DC analysis.
vSource_ac	**Voltage_source*** (AC)	Add the value of AC voltage sources into RHS column during AC analysis.
Nodes	**Node***	A vector containing all Node objects

DC_Nodes	**int**	Loop through vector **Nodes**, add all different **Nodes**[i]->**get_node_dc()** into the vector **DC_Nodes**. This allows us to organize the sequence of nodes in Jacobian matrix for DC analysis.
AC_Nodes	**int**	Loop through vector **Nodes**, add all different **Nodes**[i]->**get_node_ac()** into the vector **AC_Nodes**. This allows us to organize the sequence of nodes in conductance matrix for AC analysis.

Table 3: List of vectors built and their functions

4.8 Initial Guess for Newton-Raphson Method

For Newton-Raphson method, if the initial guess chosen is too far from the actual value, it might not be able to converge to the correct solution. Therefore, in order to make the convergence of Newton-Raphson method quick and accurate, the setting of the initial guess is essential. From the background knowledge, we know that generally, NPN BJT is operating under $V_{BE} = 0.7V$; PNP BJT is operating under $V_{EB} = 0.7V$; Diode is operating under $V_d = V_a - V_c = 0.7V$; N-MOSFET is operating under $V_{GS} > V_t = -2V$; P-MOSFET is operating under $V_{GS} < V_t = 2V$. Therefore, we make:

- $V_B = 0.7V$, $V_E = 0V$ for NPN BJT
- $V_B = 0.2V$, $V_E = 0.9V$ for PNP BJT
- $V_a = 0.7V$, $V_c = 0V$ for Diode
- $V_G = -2V$, $V_S = 0V$ for N-MOSFET
- $V_G = 2V$, $V_S = 0V$ for P-MOSFET

, as our initial guess.

4.9 Build Jacobian Matrix for DC Analysis

For DC analysis, in the Jacobian matrix, the followings are placed into the corresponding row-column location:

- conductance of normal resistors
- conductance of equivalent resistors in non-linear device models
- transconductance of voltage-controlled current sources in non-linear device models

The principles and reasons for the construction of Jacobian matrix are explained in Section 3.2.

The main process of building the Jacobian matrixes is shown in the flowchart below.

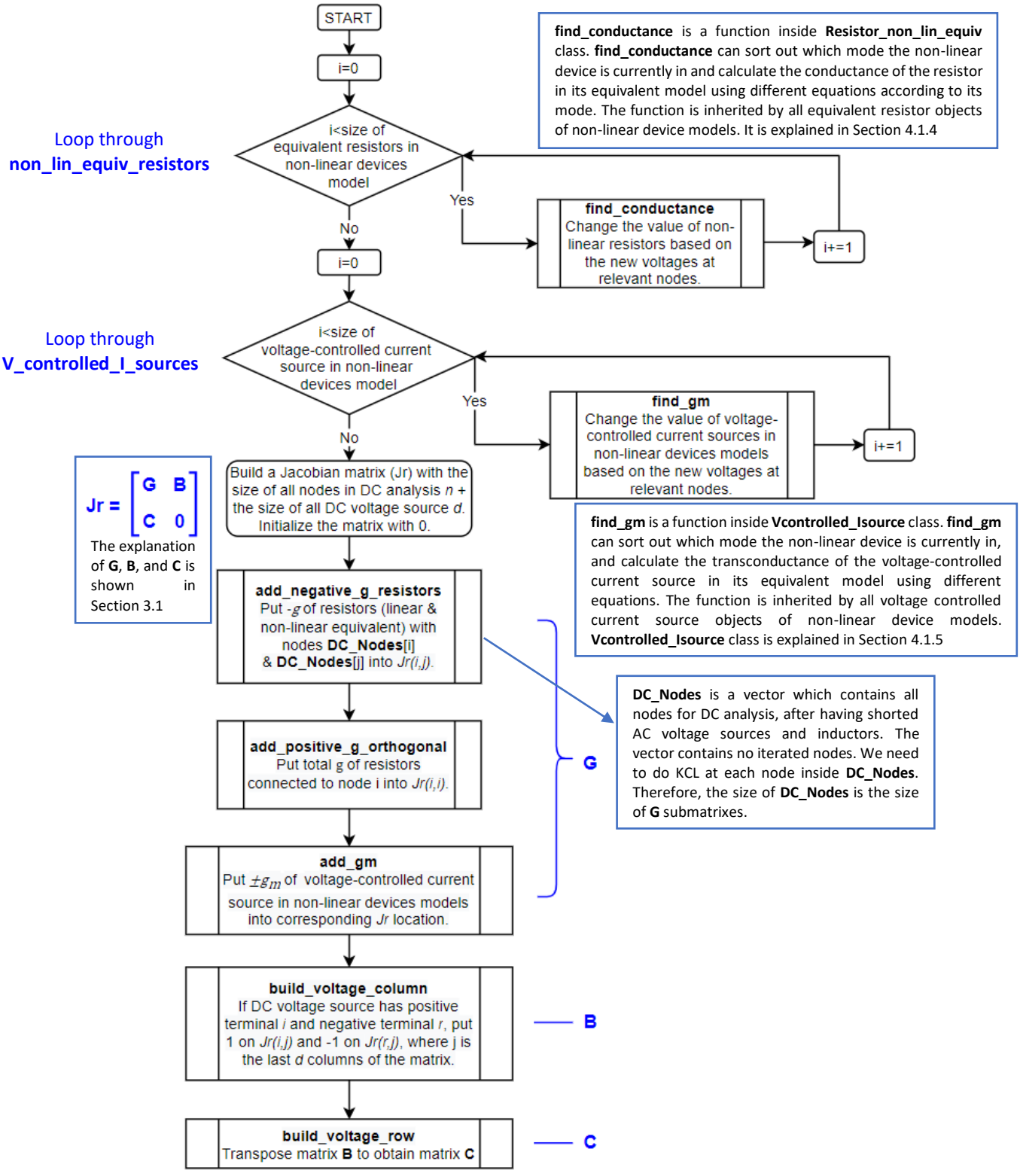


Figure 18: Flowchart for building Jacobian matrix

How functions **add_negative_g_resistors**, **add_positive_g_orthogonal**, **add_gm**, and **build_voltage_column** work is shown in the flowcharts below respectively. **build_voltage_row** is to build submatrix $\boldsymbol{C}$, which can be simply accomplished by transposing matrix $\boldsymbol{B}$. Thus it is not shown in detail anymore.

4.9.1 add_negative_g_resistors

4.9.1.1 Design

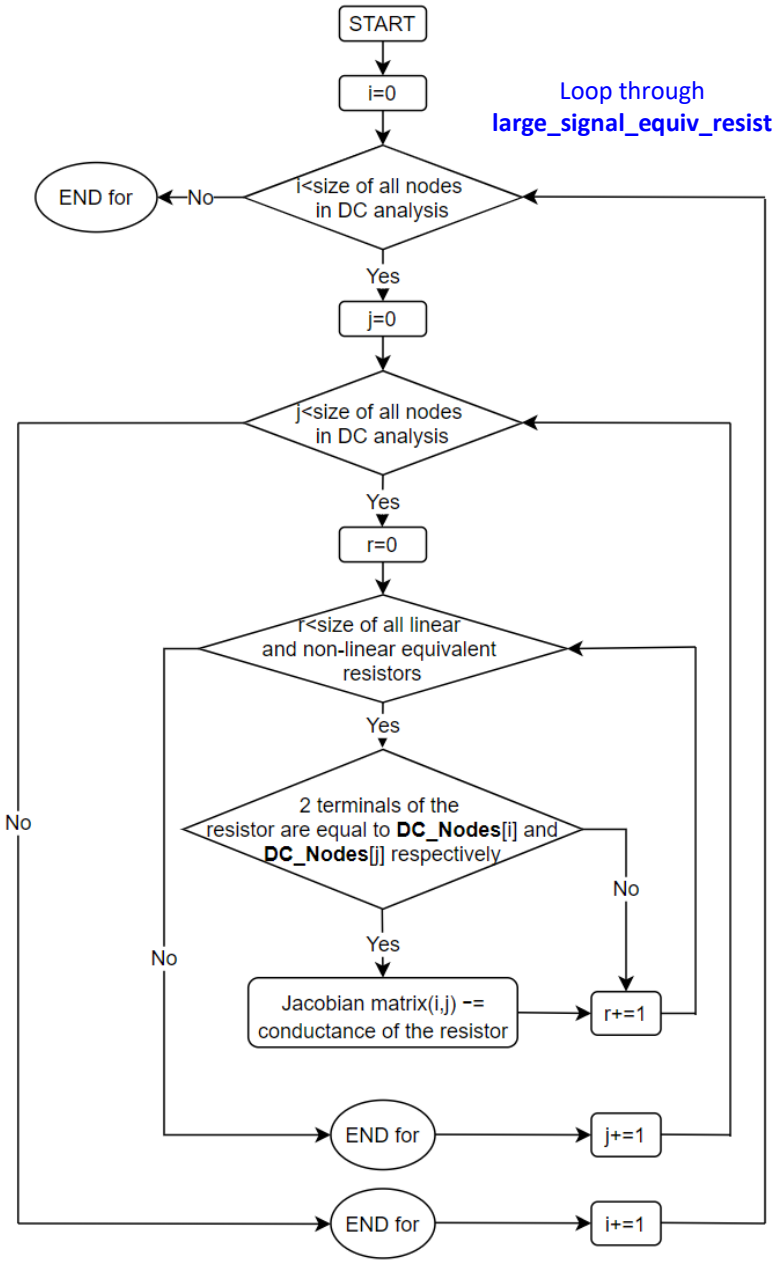


Figure 19: Flowchart for add_negative_g_resistors

4.9.1.2 Testing

Function	Expectation	Problems & Solutions	Examples
add_negative_g_resistors	Except for the diagonal, each entry (i, j) of matrix is occupied with negative conductance of all resistors directly	Problem: Initially only considered the case where **get_node1()** = i and **get_node2()** = j, and put the negative conductance into matrix (i, j) and (j, i).	R_1(N1, N2), R_2(N2, N3) Result:

	connected to **DC_Nodes**[i] and **DC_Nodes**[j].	<u>Solution:</u> Consider both cases where **get_node1()** = i, **get_node2()** = j; and **get_node1()** = j, **get_node2()** = i.	$\begin{pmatrix} 0 & -\frac{1}{R_1} & 0 \\ -\frac{1}{R_1} & 0 & -\frac{1}{R_2} \\ 0 & -\frac{1}{R_2} & 0 \end{pmatrix}$

4.9.2 add_positive_g_orthogonal

4.9.2.1 Design

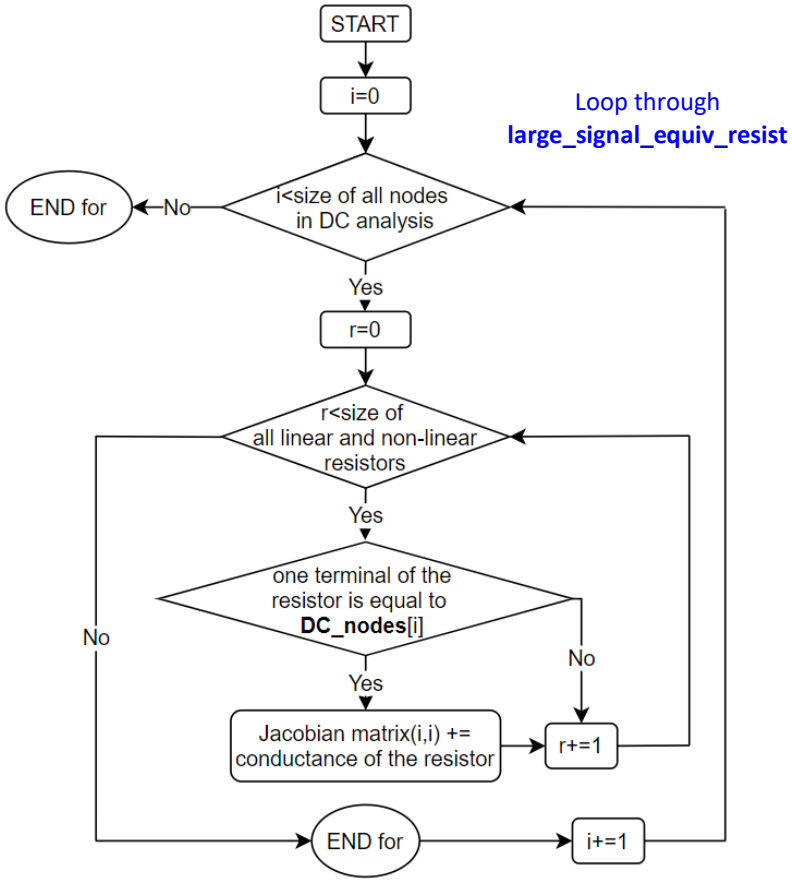


Figure 20: Flowchart for add_positive_g_orthogonal

4.9.2.2 Testing

Function	Expectation	Problems & Solutions	Examples
add_positive_g_ orthogonal	Diagonal entries of matrix are occupied with the sum of positive conductance connected to that node.	<u>Problem:</u> Initially only considered the case where **get_node1()** of the resistor is equal to i, and add the positive conductance into matrix (i, j). <u>Solution:</u> Consider both cases where **get_node1()** = i, **get_node2()** = i.	R_1(N1, N2), R_2(N2, N3) <u>Result:</u> $\begin{pmatrix} \frac{1}{R_1} & 0 & 0 \\ 0 & \frac{1}{R_2}+\frac{1}{R_1} & 0 \\ 0 & 0 & \frac{1}{R_2} \end{pmatrix}$

4.9.3 add_gm

4.9.3.1 Design

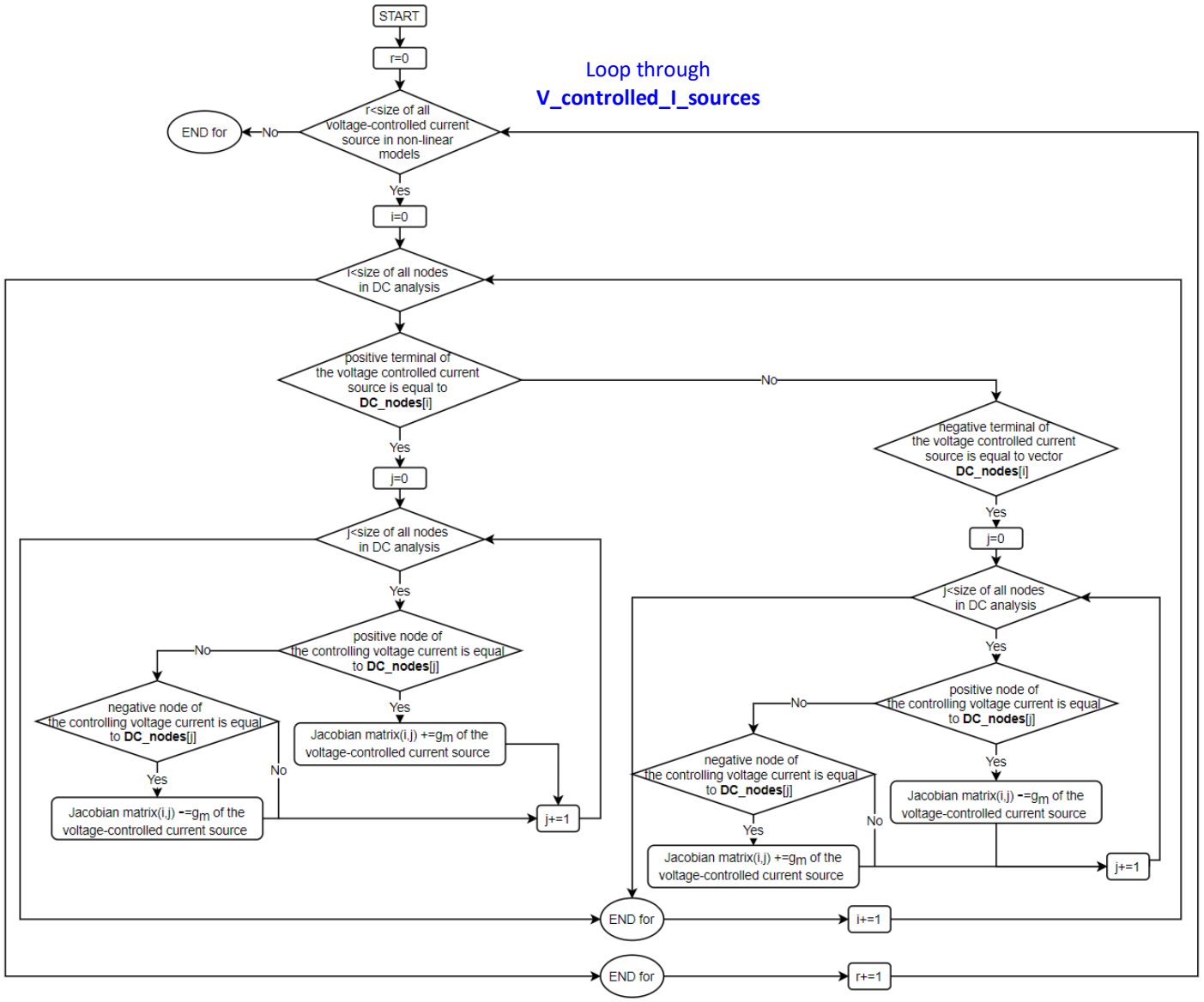


Figure 21: Flowchart for add_gm

4.9.3.2 Testing

Function	Expectation	Problems & Solutions	Examples
add_gm	Transconductance of equivalent voltage-controlled current source in non-linear device models are added to entries of the Jacobian matrix.	N/A	g_m with positive terminal N1, negative terminal N2, positive controlling node N3, and negative controlling node N2. Result: $\begin{pmatrix} 0 & -g_m & g_m \\ 0 & g_m & -g_m \\ 0 & 0 & 0 \end{pmatrix}$

4.9.4 build_voltage_column

4.9.4.1 Design

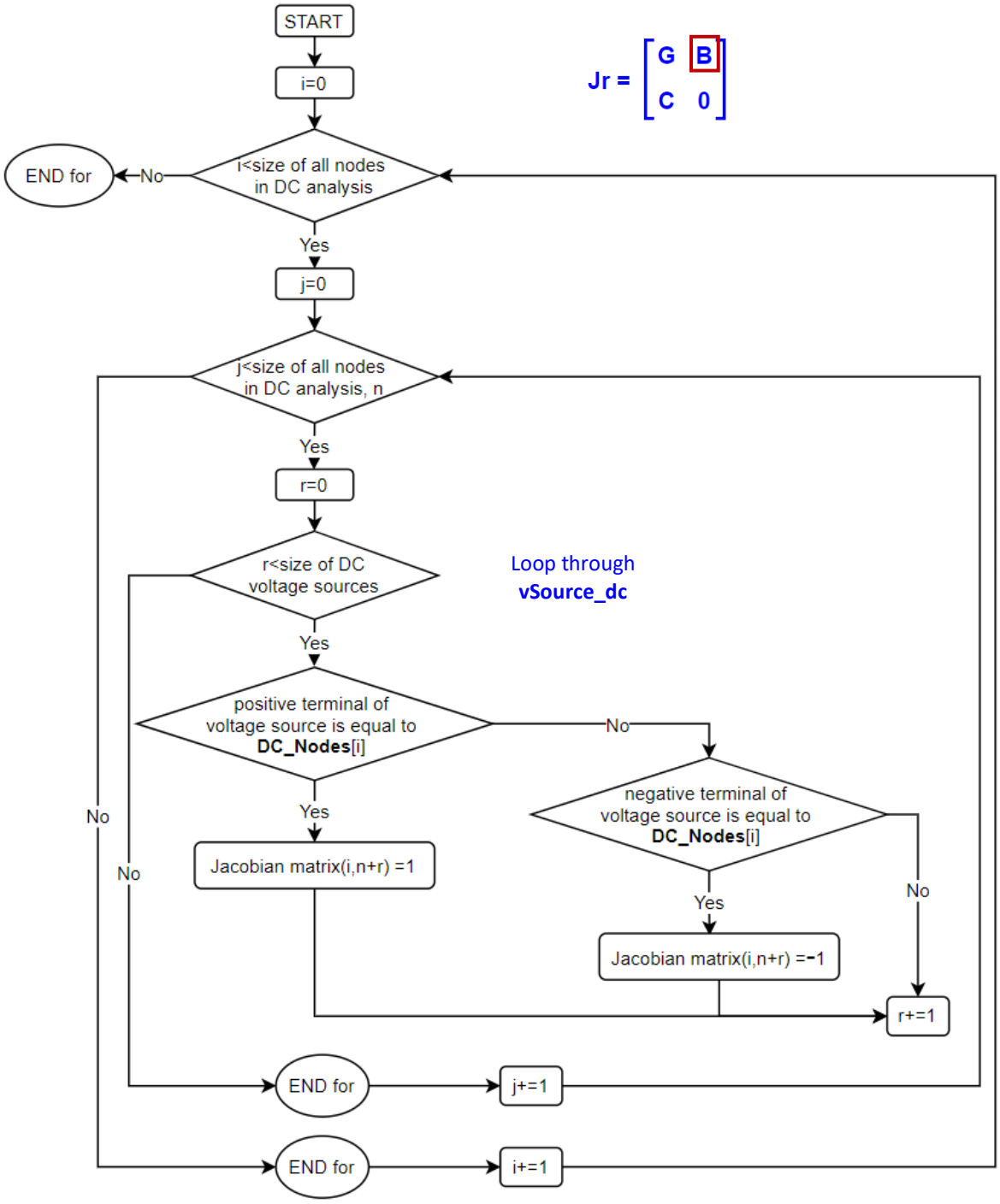


Figure 22: Flowchart for build_voltage_column

4.9.4.2 Testing

Function	Expectation	Problems & Solutions	Examples

build_voltage_column	Build submatrix $\boldsymbol{C}$. 1 and -1 are added to the last m columns of the Jacobian matrix if there are m voltage sources. If the positive terminal of the voltage source is i, 1 is put to row i. If the negative terminal of the voltage source is i, -1 is put to row i.	N/A	V_{source_1} with positive terminal N1, negative terminal N2. Result: last column of the Jacobian matrix $\begin{pmatrix} 1 \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

4.10 Build RHS Column for DC Analysis

According to the explanation of Newton-Raphson method in Section 3.2, and MNA in Section 3.1, the RHS column for DC analysis should include the value of current sources in non-linear device models, constant DC current sources, and DC voltage sources. The main process of building RHS column used in Newton-Raphson method for DC analysis is shown in the flowchart below.

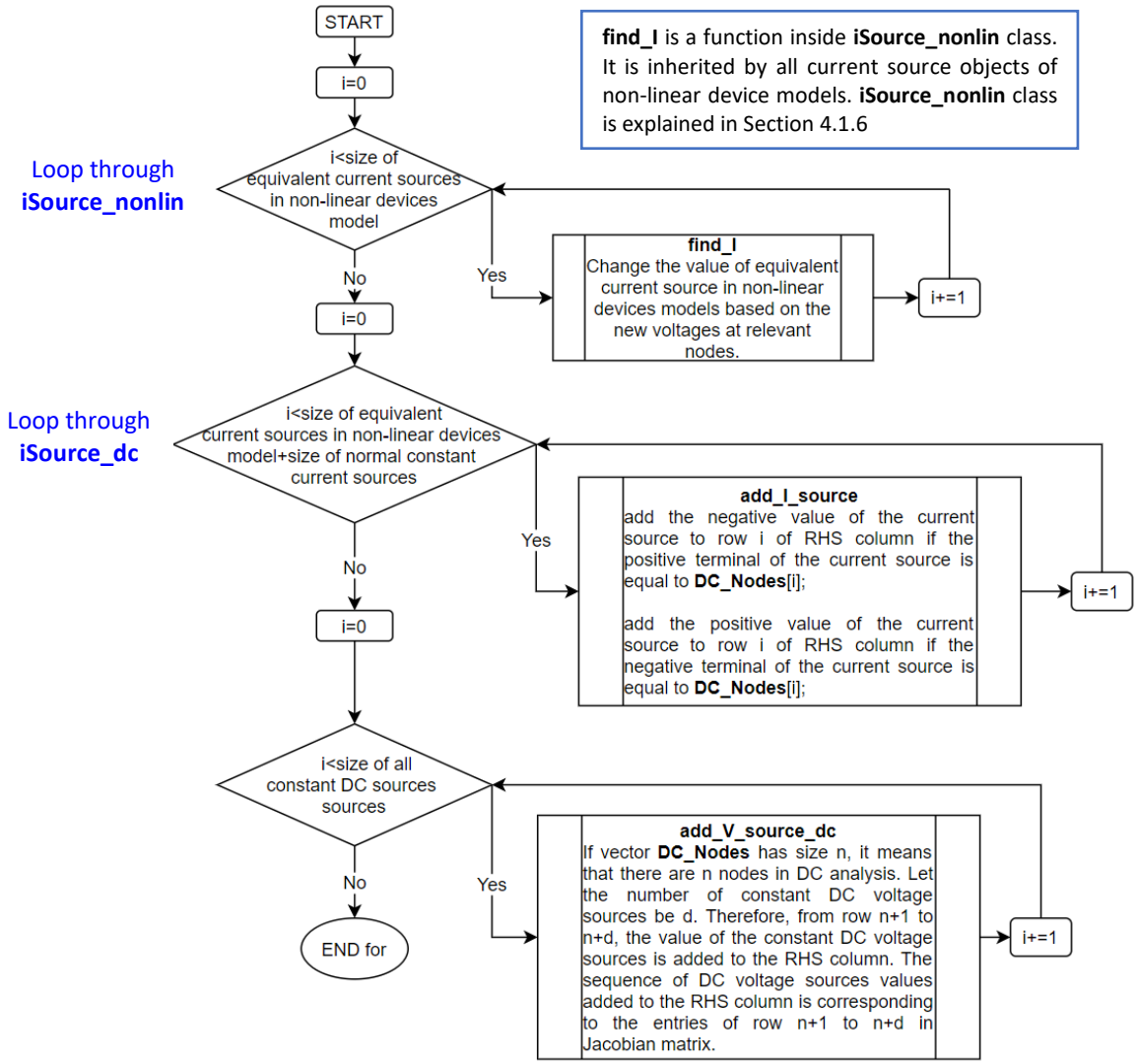


Figure 23: Flowchart for building RHS column

4.10.1add_I_source

4.10.1.1 Design

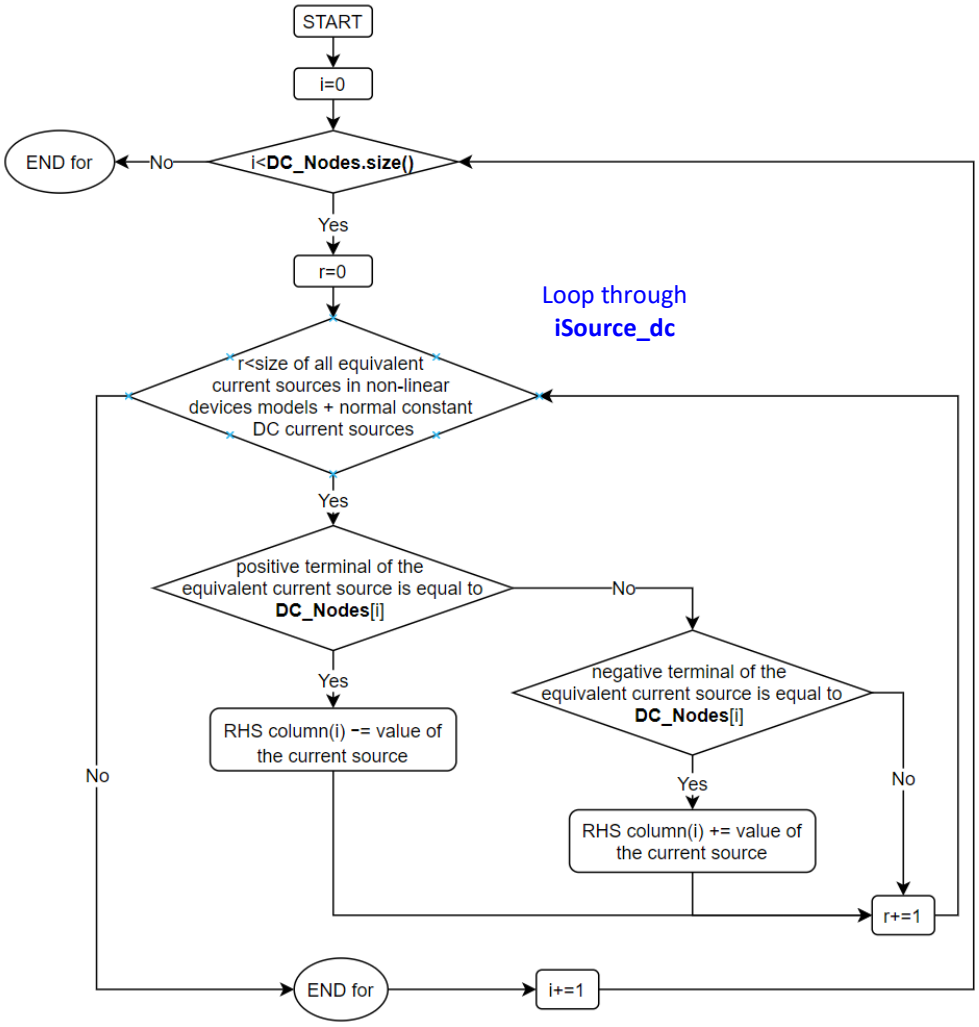


Figure 24: Flowchart for add_I_source

4.10.1.2 Testing

Function	Expectation	Problems & Solutions	Examples
add_I_source	Add negative valueS of current source to row i of RHS column if the positive terminal of the current source is **DC_Nodes**[i]. Add positive value of current source to row i of RHS column if the negative terminal of the current source is **DC_Nodes**[i].	Problem: Previously we treated current sources in non-linear device models and normal DC current sources differently and added them to RHS column separately. This requires one more time of for loop, and is less efficient. Solution: Put current sources in non-linear device models and normal DC current sources into one vector and add them into RHS column together.	A current source with value I_{source_1} , whose positive terminal is N1, and whose negative terminal is N2 Result: RHS column is $\begin{pmatrix} -I_{source_1} \\ I_{source_1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

4.10.2add_V_source_dc

4.10.2.1 Design

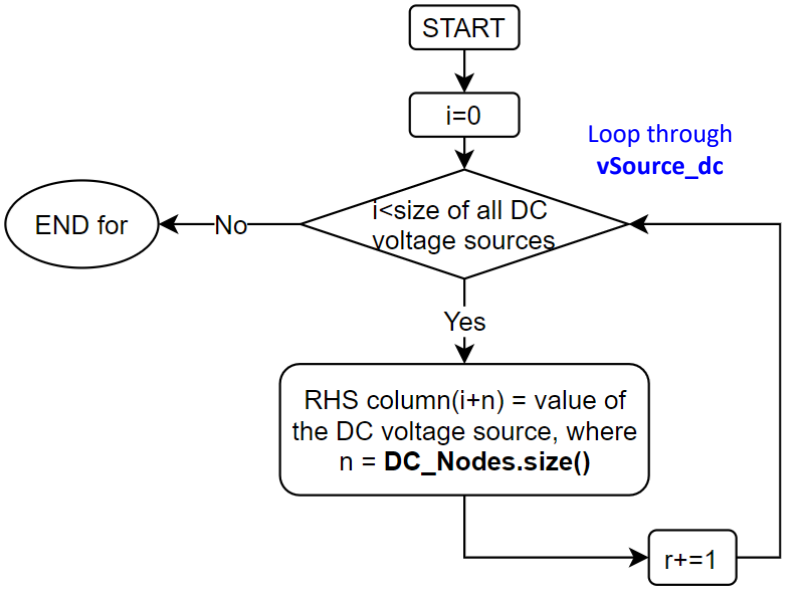


Figure 25: Flowchart for add_V_source_dc

4.10.2.2 Testing

Function	Expectation	Problems & Solutions	Examples
add_V_source_dc	Add value of voltage source to last d rows of RHS column if there are d DC voltage sources.	N/A	A DC voltage source with value V_{source_1}. Result: RHS column is $\begin{pmatrix} 0 \\ 0 \\ \vdots \\ V_{source_1} \end{pmatrix}$

4.11 Newton-Raphson Method to Find DC Operating Point

4.11.1 Design

The flowchart for using Newton-Raphson method to find DC operating point is shown below.

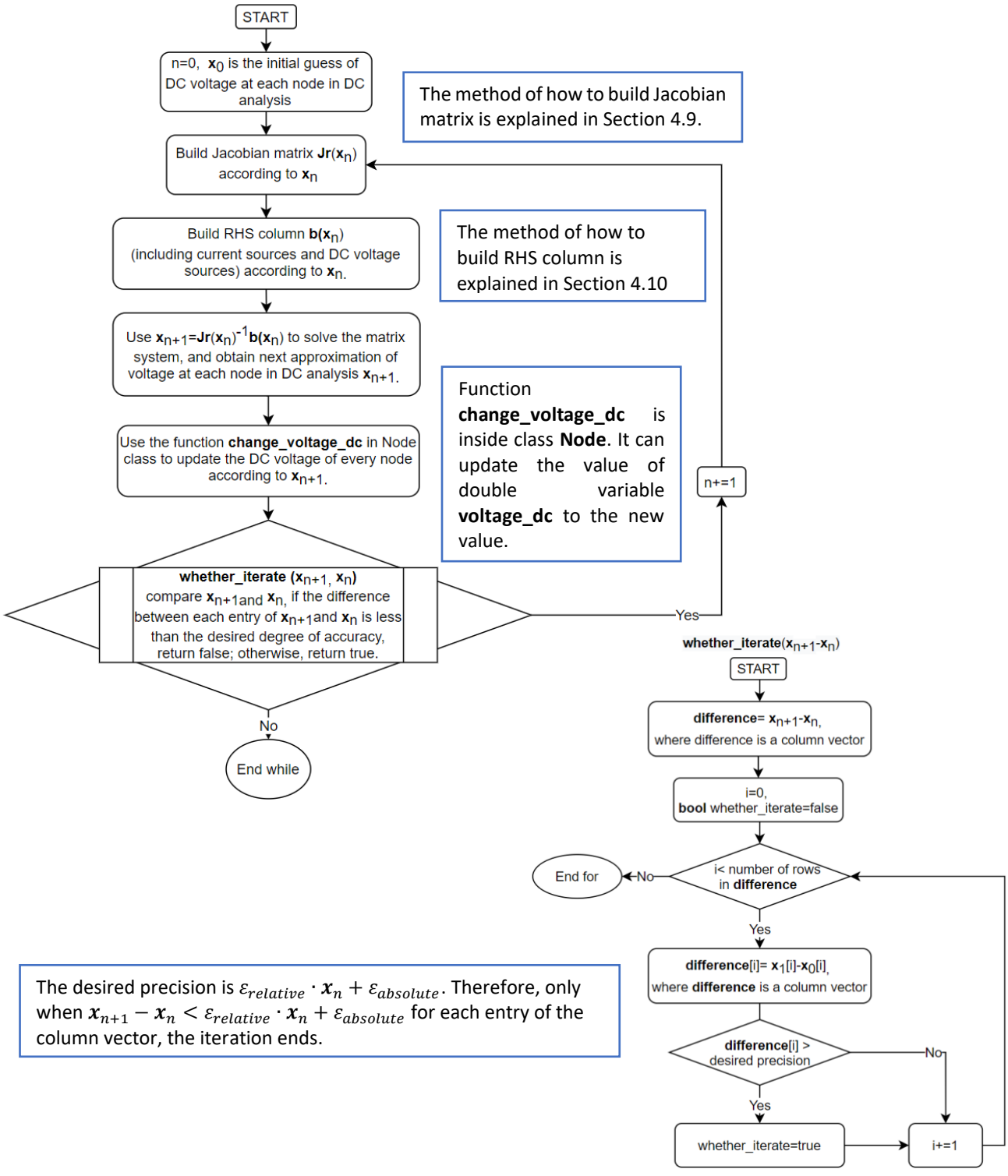


Figure 26: Flowchart for Newton-Raphson method to find DC operating point

In our program, we set $\varepsilon_{relative} = 0.01$, and $\varepsilon_{absolute} = 0.01$. This is because if $\varepsilon_{relative}$ and $\varepsilon_{absolute}$ are too large, the iteration will stop too fast and the convergence might not be reached. If $\varepsilon_{relative}$ and $\varepsilon_{absolute}$ are too small, the program will iterate too many times with useless precision, and it wastes memory space and is time-consuming [6]. 0.01 is a value that can reach a balance between precision and efficiency.

It is worth noting that our program can reidentify the operating mode of the non-linear device at each iteration and use different equations corresponding to different modes to calculate the values of resistors, voltage-controlled current sources, and current sources in their equivalent model. This makes our program more accurate and manages to converge to the correct operating mode for non-linear devices.

After doing Newton-Raphson iteration and update the value of **voltage_dc** in every **Node** object using function **change_voltage_dc**, the value of DC operating voltage at each node is recorded down. DC analysis is then finished.

4.11.2 Testing

Function	Expectation	Problems & Solutions	Examples
Build Jacobian matrix	The Jacobian matrix with conductance and transconductance values placed at correct locations is built.	Shown in Section 4.9.	V1 N001 0 5 V2 N004 0 AC 1 0 Q1 N003 N002 0 NPN C1 N002 N004 100n R2 N005 N002 90k R3 N003 N002 220k R4 N001 N003 10k R1 N005 0 5k Result: Jacobian matrix is $\begin{pmatrix} \frac{1}{R_1}+\frac{1}{R_2} & 0 & 0 & -\frac{1}{R_2} & 0 \\ 0 & \frac{1}{R_4}+\frac{1}{R_3}+g_{out} & -\frac{1}{R_4} & g_m & 0 \\ 0 & -\frac{1}{R_4} & \frac{1}{R_4} & 0 & 1 \\ -\frac{1}{R_2} & -\frac{1}{R_3} & 0 & \frac{1}{R_2}+\frac{1}{R_3}+g_{in} & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$
Build RHS column	The RHS column with the values of I_{source} and V_{source} added to the correct place is built.	Shown in Section 4.10.	Same example continues. Result: RHS column is $\begin{pmatrix} 0 \\ -I_{CE} \\ -I_{BE} \\ 0 \\ 5 \end{pmatrix}$.
whether_iterate	Newton Raphson iteration stops at correct precision level: $\boldsymbol{x}_{n+1} - \boldsymbol{x}_n < 0.01\boldsymbol{x}_n + 0.01$	N/A	The same example continues. Result: barrier: (0.0103312, 0) difference: (0.000922042, 0) barrier: (0.0336842, 0) difference: (0.0528751, 0) barrier: (0.06, 0) difference: (0, 0) barrier: (0.0162922, 0) difference: (0.0175188, 0) barrier: (0.00999737, 0) difference: (5.28751e-06, 0) barrier represents $0.01\boldsymbol{x}_n + 0.01$; difference represents $\boldsymbol{x}_{n+1} - \boldsymbol{x}_n$ For every node, the difference is less than barrier.

4.12 AC Analysis

To find out the frequency response of the circuit, we need to loop through the frequency vector and do AC analysis at each frequency. Before doing AC analysis, we need to loop through **non_lin_equiv_resistors** and **V_controlled_I_sources** to find the conductance/ transconductance of equivalent resistors/ voltage-controlled current sources of non-linear device models under DC operating points.

For AC analysis, we need to build the conductance matrix. Firstly, the conductance of objects contained in vector **ssem_equiv_resistors** is added to the correct row-column location of the matrix. The method is shown in Section 4.9.1 and Section 4.9.2. Then, the transconductance of voltage-controlled current sources contained in vector **V_controlled_I_sources** have added to the matrix using the method shown in Section 4.9.3. Finally, for the last few rows and columns of the matrix which reflect the relations between voltages at 2 terminals of AC voltage sources, we need to loop through vector **voltage_ac** to add correct terms into the matrix using the method explained in Section 4.9.4.

RHS column for AC analysis is built by firstly looping through vector **current_ac** to add total current sources flowing into each node into the column. The overview of the procedure is shown in Section 4.10. Secondly, for the last few entries of the column, we loop through vector **voltage_ac** to add values of AC voltage sources into the column. The method used is the same as Section 4.10.2.

By solving the matrix system, we can obtain the exact voltage at each node in the circuit. Thus, the transfer function between the input and output node can be calculated by dividing the voltage at these two nodes.

4.13 Output

4.13.1 Design

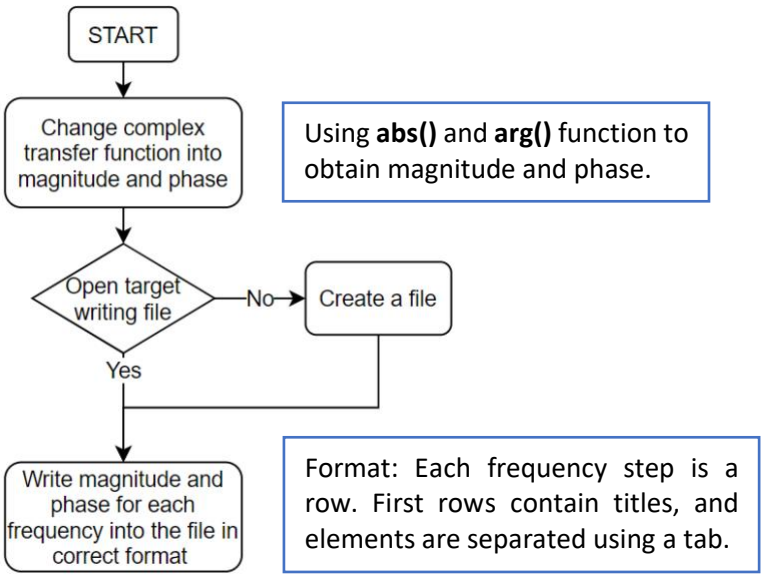


Figure 27: Flowchart for output file

4.13.2 Testing

Function	Expectation	Problems & Solutions	Example
Write frequencies and corresponding magnitudes and phases into txt file	Frequencies and corresponding magnitudes and phases are written into the file in the correct format.	Problem: The code in MATLAB does not accept space between data Solution: Change space to tab	Frequency Magnitude Phase 10 0.0110385 89.3675 20 0.022073 88.7352
Change complex voltage output to magnitude and phase	Correct magnitudes and phases for each frequency are printed	N/A	(3,4) to magnitude 5, phase 53.13

5 Problem Solving

5.1 Create a New Class for Node

Problems:

Initially, nodes were all in the type of integer, and this caused the following problems.

Problem 1:

To implement short circuit and open circuit, we need to change the nodes connected to some components. Therefore, all component classes are required to have functions to change the connected nodes. However, it is inefficient since similar functions are written repeatedly for each class.

Problem 2:

For short circuit and open circuit, we need to loop through all different types of component vectors separately to find the specified node and change it to the new node. The code is redundant which needs to be improved.

Problem 3:

After short-circuiting, the total number of nodes may be reduced, but it's hard to find out which node(s) have been omitted in the equivalent circuit.

Problem 4:

Different components are shorted for DC and AC. The change of nodes in DC analysis means that the nodes' original values cannot be restored, and the information of original nodes is lost. Then the following AC analysis cannot be run correctly. We need a way to hold the original values of the nodes.

Solutions:

The problems above can get solved altogether by creating a new class for nodes. The **Node** class can be used as attributes in the circuit-component classes.

For example, for the resistor class:

- Before: **Resistor(double resistance, int n1, int n2)**
- After: **Resistor(double resistance, Node* n1, Node* n2)**

Instead of having functions for each component class to change the connected nodes, we now only need these change-node functions in the **Node** class. Since all other components have pointers to nodes as their attributes, changes made in **Node** objects can be reflected automatically in circuit-component objects. This solves problem 1.

All the objects of the class **Node** can be stored together in a **Node*** vector. When implementing short and open circuit, we only need to loop through the **Node*** vector to change the nodes. This solves problem 2.

Since all pointers to nodes are stored in a vector, the omitted node(s) can easily be found using the vector. This solves problem 3.

To solve problem 4, we've added 2 attributes to the node class to store the node value of DC and AC, so that the original value of the node can be retained.

The details of implementation are as shown below:

```cpp
class Node {
public:
    Node(int n) : node_orig(n) {
        node_dc = n;
        node_ac = n;
        voltage_dc = 0;
        voltage_ac = 0;
    }

    int get_node_orig() const { return node_orig; }
    int get_node_dc() const { return node_dc; }
    int get_node_ac() const { return node_ac; }
    std::complex<double> get_voltage_dc() const { return voltage_dc; }
    std::complex<double> get_voltage_ac() const { return voltage_ac; }

    void change_node_dc(int new_node) { node_dc = new_node; }
    void change_node_ac(int new_node) { node_ac = new_node; }
    void change_voltage_dc(std::complex<double> voltage) { voltage_dc = voltage; }
    void change_voltage_ac(std::complex<double> voltage) { voltage_ac = voltage; }

private:
    int node_orig;
    int node_dc;
    int node_ac;
    std::complex<double> voltage_dc;
    std::complex<double> voltage_ac;
};
```

Figure 28: Code for class Node

5.2 Active model -> All Modes Situations

Problem:

At the beginning, we did not consider different operating modes of non-linear components and assumed DC operating points. But after starting to do DC analysis, we realized the formulas for equivalent components are different in each operating mode.

Solution:

Functions are made to identify the operating modes of components (Section 4.2). Then in the classes for equivalent components, different formulae are used for calculation in different modes (Section 4.1).

5.3 Place Formulae and Constants into Class

Problem:

For processes like DC, AC analysis, equivalent values are required to be calculated. As a result, there are several repeated formulae in different functions for each component. This lowered the efficiency.

Another problem was values like V_A and k are set as global values, but these values can be different for different components or different types.

Solution:

We decided to add these formulae into classes, functions in classes receive values and return equivalent results. These functions can be called several times in different places, and calculation results are stored in objects (Section 0).

Values like V_A and β are stored in the corresponding classes. For components that have more than one type like BJT, the value return will depend on the type.

5.4 Add PNP BJT

Problem:

PNP BJT was not considered during the analysis. So, if files contain PNP, there will be an error.

Solution:

in class BJT, parameters for PNP are added. The formulae of equivalent values for PNP BJT are different from NPN BJT. It is changed by replacing all V_{CE} by V_{EC} and replacing all V_{BE} by V_{EB}. Also, the positive node and negative node of Voltage-controlled current sources and current sources are reversed.

5.5 Use Pointers Instead of Objects

Problem:

Initially, our program passes the object itself into functions. However, when passing an object into a function, the function won't be able to update the actual value of the passed object in memory. In our program, an object of component is stored in several vectors (e.g., non-linear equivalent resistors are stored in both vectors **ssem_equiv_resistors** & **non-lin-equiv-resistors**), so the objects need to be changed in several different processes/locations in the program.

Solution:

To solve the problem, we use pointers to objects instead. When we pass the object pointer into a function, the function has access to that specific object, not a copy of it, because what we are passing is the address of the object in memory. This allows us to refer to the same space in memory from multiple locations in the program. A change to the object in one location can be reflected in other locations.

6 Testing

The testing examples chosen are intended to cover as many situations as possible to test for the usability of our program.

6.1 Case 1: NPN BJT in Active Mode

The reason why we choose an NPN type BJT in active mode as the first testing case is that active mode is the most common mode for BJT. Therefore, we need to make sure our written program works well for BJT in active mode first. The circuit diagram and .txt simulation file is shown below.

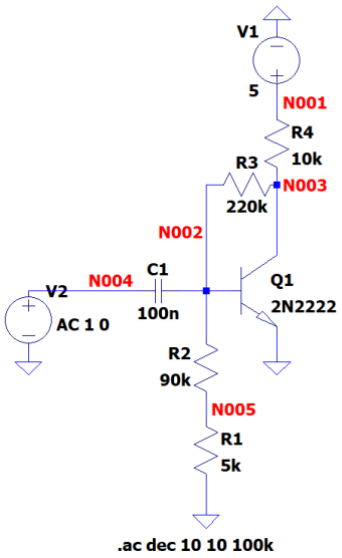


Figure 29: Circuit diagram for case 1

```
V1 N001 0 5
V2 N004 0 AC 1 0
Q1 N003 N002 0 NPN
C1 N002 N004 100n
R2 N005 N002 90k
R3 N003 N002 220k
R4 N001 N003 10k
R1 N005 0 5k
.ac dec 10 10 100k
.end
```

Figure 30: TXT file for circuit simulation for case 1

BJT 2N2222 model is chosen in LTSpice, because 2N2222 model has the same parameter values (V_A, β_f, β_r …) as the values that we used in our written program. This is explained in detail in Section 2.2.2

For DC analysis, to check whether the Jacobian matrix and RHS matrix built are correct, we draw out the equivalent circuit for the non-linear device BJT using the principles explained in Section 3.2.

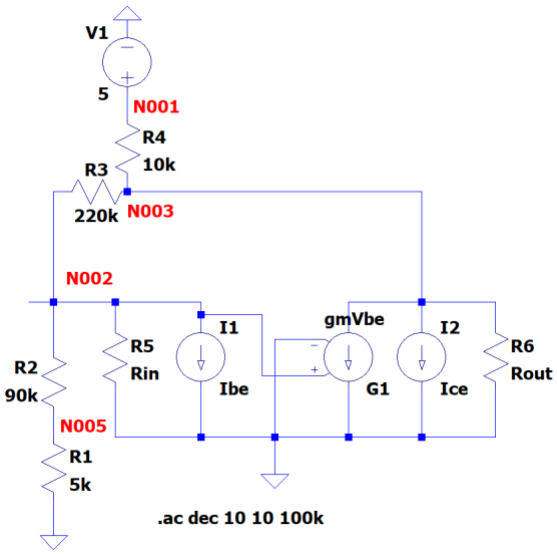


Figure 31: Equivalent diagram for case 1 DC Analysis

We write out the KCL equation at each node:

N001: $\frac{V_1}{R4} - \frac{V_3}{R4} = 0$

N003: $\frac{V_3}{R_4} - \frac{V_1}{R_4} + \frac{V_3}{R_3} - \frac{V_2}{R_3} + g_m V_2 + \frac{V_3}{R_{out}} = -I_{CE}$

N002: $\frac{V_2}{R_2} - \frac{V_5}{R_2} - \frac{V_3}{R_3} + \frac{V_2}{R_3} + \frac{V_2}{R_{in}} = -I_{BE}$

N005: $\frac{V_5}{R_2} - \frac{V_2}{R_2} + \frac{V_5}{R_1} = 0$

$V_1 = 5$

Therefore, the Jacobian matrix $= \begin{pmatrix} \frac{1}{R_1} + \frac{1}{R_2} & 0 & 0 & -\frac{1}{R_2} & 0 \\ 0 & \frac{1}{R_4} + \frac{1}{R_3} + g_{out} & -\frac{1}{R_4} & g_m & 0 \\ 0 & -\frac{1}{R_4} & \frac{1}{R_4} & 0 & 1 \\ -\frac{1}{R_2} & -\frac{1}{R_3} & 0 & \frac{1}{R_2} + \frac{1}{R_3} + g_{in} & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$, and the RHS column is $\begin{pmatrix} 0 \\ -I_{CE} \\ -I_{BE} \\ 0 \\ 5 \end{pmatrix}$.

We use $V_{BE} = 0.7V$, $V_{CE} = 0.7V$ as our initial guess for Newton-Raphson method. This means that we set $V_2 = 0.7V$, $V_3 = 0.7V$, and voltage at all other nodes to be equal to $0V$ so as to obtain the initial value for g_m, R_{in}, and R_{out}. The method of how to calculate g_m, R_{in}, and R_{out} is shown in Section 3.2.2.

The first Jacobian matrix and RHS column obtained from our program is shown below:

Initial Jacobian Matrix					Initial RHS Column
(0.000211111, 0)	(0, 0)	(0, 0)	(-1.11111e-05, 0)	(0, 0)	(0, 0)
(0, 0)	(0.000249171, 0)	(-0.0001, 0)	(0.582548, 0)	(0, 0)	(0.393324, 0)
(0, 0)	(-0.0001, 0)	(0.0001, 0)	(0, 0)	(1, 0)	(0, 0)
(-1.11111e-05, 0)	(-4.54545e-06, 0)	(0, 0)	(0.00292842, 0)	(0, 0)	(0.00196611, 0)
(0, 0)	(0, 0)	(1, 0)	(0, 0)	(0, 0)	(5, 0)

Table 4: Screenshot for initial Jacobian matrix and RHS column

After substituting the initial guess $V_2 = 0.7V$, $V_3 = 0.7V$, $V_1 = 0V$, $V_4 = 0.7V$, and the value of R_1, R_2, R_3, R_4 into the KCL equation sets and the Jacobian listed above, we can confirm that the Jacobian matrix and RHS column obtained by our program is correct.

The program solves the matrix equation system and obtains the next approximation of voltage at each node.

First Vx Solution: Vx(1)	
(0.0355296, 0)	N001: (5, -0)
(2.27857, 0)	N000: (0, 0)
(5, -0)	N004: (0, 0)
(0.675063, 0)	N003: (2.27857, 0)
(-0.000272143, -0)	N002: (0.675063, 0)
	N005: (0.0355296, 0)

Table 5 Screenshot for initial matrix system solution

Since $V_{BE} = V_2 = 0.67V > 0V$, and $V_{CB} = V_{32} = 2.28V - 0.68V = 1.6V > 0V$, BJT is still in active mode. The following chart shows the iteration process:

Jacobian Matrix					RHS Column	Vx Solution
(0.000211111, 0)	(0, 0)	(0, 0)	(-1.11111e-05, 0)	(0, 0)	(0, 0)	(0.0342721, 0)
(0, 0)	(0.000157884, 0)	(-0.0001, 0)	(0.218211, 0)	(0, 0)	(0.141975, 0)	(2.42494, 0)
(0, 0)	(-0.0001, 0)	(0.0001, 0)	(0, 0)	(1, 0)	(0, 0)	(5, -0)
(-1.11111e-05, 0)	(-4.54545e-06, 0)	(0, 0)	(0.00110673, 0)	(0, 0)	(0.000709269, 0)	(0.65117, 0)
(0, 0)	(0, 0)	(1, 0)	(0, 0)	(0, 0)	(5, 0)	(-0.000257506, -0)

Jacobian Matrix					RHS Column	Vx Solution
(0.000211111, 0)	(0, 0)	(0, 0)	(−1.11111e−05, 0)	(0, 0)	(0, 0)	(0.0331169, 0)
(0, 0)	(0.000125056, 0)	(−0.0001, 0)	(0.0840282, 0)	(0, 0)	(0.0526685, 0)	(2.36842, 0)
(0, 0)	(−0.0001, 0)	(0.0001, 0)	(0, 0)	(1, 0)	(0, 0)	(5, −0)
(−1.11111e−05, 0)	(−4.54545e−06, 0)	(0, 0)	(0.00043582, 0)	(0, 0)	(0.000263094, 0)	(0.629221, 0)
(0, 0)	(0, 0)	(1, 0)	(0, 0)	(0, 0)	(5, 0)	(−0.000263158, −0)
Jacobian Matrix					RHS Column	Vx Solution
(0.000211111, 0)	(0, 0)	(0, 0)	(−1.11111e−05, 0)	(0, 0)	(0, 0)	(0.0321949, 0)
(0, 0)	(0.00011307, 0)	(−0.0001, 0)	(0.0349032, 0)	(0, 0)	(0.0211122, 0)	(2.31554, 0)
(0, 0)	(−0.0001, 0)	(0.0001, 0)	(0, 0)	(1, 0)	(0, 0)	(5, −0)
(−1.11111e−05, 0)	(−4.54545e−06, 0)	(0, 0)	(0.000190195, 0)	(0, 0)	(0.00010546, 0)	(0.611703, 0)
(0, 0)	(0, 0)	(1, 0)	(0, 0)	(0, 0)	(5, 0)	(−0.000268446, −0)
Jacobian Matrix					RHS Column	Vx Solution
(0.000211111, 0)	(0, 0)	(0, 0)	(−1.11111e−05, 0)	(0, 0)	(0, 0)	(0.0316812, 0)
(0, 0)	(0.000108776, 0)	(−0.0001, 0)	(0.0173081, 0)	(0, 0)	(0.0101672, 0)	(2.28616, 0)
(0, 0)	(−0.0001, 0)	(0.0001, 0)	(0, 0)	(1, 0)	(0, 0)	(5, −0)
(−1.11111e−05, 0)	(−4.54545e−06, 0)	(0, 0)	(0.00010222, 0)	(0, 0)	(5.0787e−05, 0)	(0.601943, 0)
(0, 0)	(0, 0)	(1, 0)	(0, 0)	(0, 0)	(5, 0)	(−0.000271384, −0)

Table 6: Screenshot for Newton-Raphson iteration process

After the desired degree of accuracy is reached, which is in our program: $\boldsymbol{x}_{n+1} - \boldsymbol{x}_n < 0.01 + 0.01\boldsymbol{x}_n$, the iteration stops. In this test case, five times of iteration are performed in total until the result converges to the final operating point.

N001: (5, −0)
N000: (0, 0)
N004: (0, 0)
N003: (2.28616, 0)
N002: (0.601943, 0)
N005: (0.0316812, 0)

Figure 32: Operating point found by our program

--- Operating Point ---

V(n001):	5	voltage
V(n004):	0	voltage
V(n003):	2.33511	voltage
V(n002):	0.619794	voltage
V(n005):	0.0326208	voltage

Figure 33: Operating point found by LTSpice

After obtaining the DC operating point, we can do the AC analysis.

The equivalent circuit for AC analysis is:

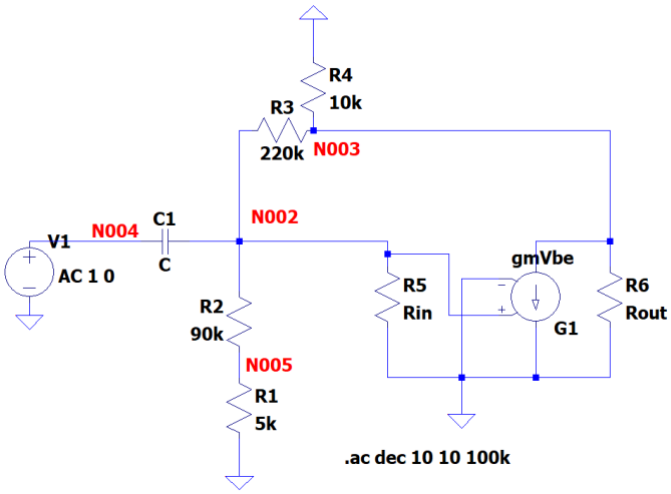


Figure 34: Equivalent circuit for case 1 AC analysis

N003: $\frac{V_3}{R_4} + \frac{V_3}{R_3} - \frac{V_2}{R_3} + g_m V_2 + \frac{V_3}{R_{out}} = 0$

N002: $\frac{V_2}{R_2} - \frac{V_5}{R_2} - \frac{V_3}{R_3} + \frac{V_2}{R_3} + \frac{V_2}{R_{in}} + j\omega C \cdot V_2 - j\omega C \cdot V_4 = 0$

N005: $\frac{V_5}{R_2} - \frac{V_2}{R_2} + \frac{V_5}{R_1} = 0$

N004: $j\omega C \cdot V_4 - j\omega C \cdot V_2 = 0$

$V_4 = 1$

Since we are doing the AC analysis in the range $f = 10Hz$ to $f = 100kHz$, we need to substitute f values inside the conductance matrix with step $\Delta f = 10Hz$ when $10Hz < f < 1000Hz$, and step $\Delta f = 1000Hz$ when $1000Hz < f < 100kHz$, and then solve the matrix system at each step.

Take $f = 10Hz$ as an example, the conductance matrix and the RHS column are:

gmatrix					vicolumn
(7.42257e-05, 6.28319e-06)	(-1.11111e-05, 0)	(-4.54545e-06, 0)	(0, -6.28319e-06)	(0, 0)	(0, 0)
(-1.11111e-05, 0)	(0.000211111, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
(0.0117093, 0)	(0, 0)	(0.000107408, 0)	(0, 0)	(0, 0)	(0, 0)
(0, -6.28319e-06)	(0, 0)	(0, 0)	(0, 6.28319e-06)	(1, 0)	(0, 0)
(0, 0)	(0, 0)	(0, 0)	(1, 0)	(0, 0)	(1, 0)

Table 7: Conductance matrix and RHS column when f=10Hz

After substituting the relevant values into the above-mentioned equations, we can confirm that the AC analysis matrix system in our program is correct. The following graph shows the solution of voltage at each node when $f = 10Hz$.

Vx	
(0.000121849, 0.0110379)	N001: (0, 0)
(6.41311e-06, 0.00058094)	N000: (0, 0)
(-0.0132836, -1.20331)	N004: (1, 1.10488e-22)
(1, 1.10488e-22)	N003: (-0.0132836, -1.20331)
(-6.93529e-08, -6.28242e-06)	N002: (0.000121849, 0.0110379)
	N005: (6.41311e-06, 0.00058094)

Table 8: AC analysis solution for f=10Hz

We take N004 as the reference node, and N002 as the output node, and stores the magnitude and the phase of $\frac{V_2}{V_4}$ for each frequency step. Then we use MATLAB to plot the graph of magnitude and phase versus frequency. The table below shows the comparison between the plot obtained by our program and LTSpice.

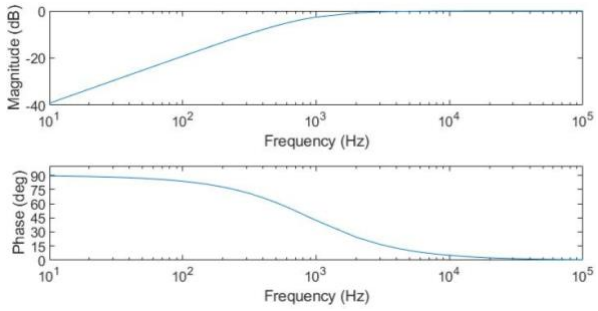


Figure 35: Frequency response plot by our program

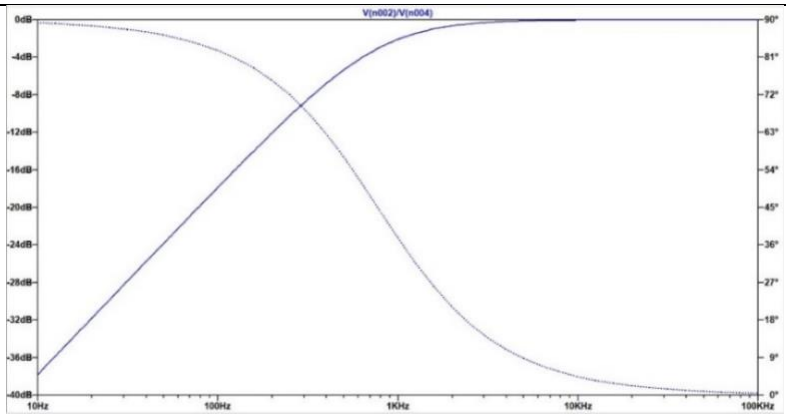


Figure 36: Frequency response plot by LTSpice

Table 9: Comparison of the result between our program and LTSpice for case 1

We can observe from the above table that our program produces the same AC analysis plot as LTSpice. Therefore, it shows that our program works for the case when BJT is in active mode.

6.2 Case 2: N-MOSFET in Saturation Mode

Saturation mode is the most common mode for MOSFET. Therefore, we need to make sure our written program works well for MOSFET in saturation mode first. The circuit diagram and .txt simulation file is shown below.

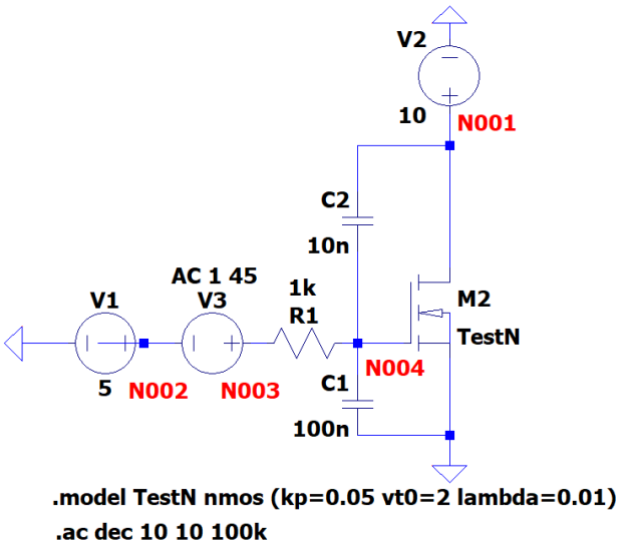


Figure 37: Circuit diagram for case 2

```
R1 N004 N003 1k
V2 N001 0 10
V3 N003 N002 AC 1 45
V1 N002 0 5
M2 N001 N004 0 NMOS
C2 N001 N004 10n
C1 N004 0 100n
.ac dec 10 10 100k
.end
```

Figure 38: TXT file for circuit simulation for case 2

We set the parameters: $k_p = 0.05, V_t = 2, V_A = \frac{1}{\lambda} = 100$ for MOSFET in LTSpice. This is the same as the parameters that we used in our program.

For DC analysis, in order to check whether the Jacobian matrix and RHS matrix built are correct, we draw out the equivalent circuit for the non-linear device MOSFET using the principles explained in Section 3.2.

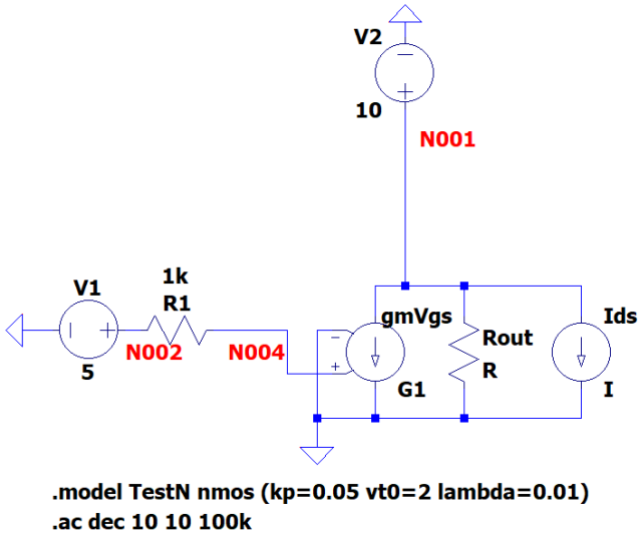


Figure 39: Equivalent circuit for case 2 DC analysis

N004: $\frac{V_4}{R_1} - \frac{V_2}{R_1} = 0$

N001: $\frac{V_1}{R_{out}} + g_m V_4 = -I_{DS}$

N002： $\frac{V_2}{R_1} - \frac{V_4}{R_1} = 0$

$V_1 = 10$

$V_2 = 5$

Therefore, the Jacobian matrix $= \begin{pmatrix} \frac{1}{R_1} & 0 & -\frac{1}{R_1} & 0 & 0 \\ g_m & \frac{1}{R_{out}} & 0 & 1 & 0 \\ -\frac{1}{R_1} & 0 & \frac{1}{R_1} & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$, and the RHS column is $\begin{pmatrix} 0 \\ -I_{DS} \\ 0 \\ 10 \\ 5 \end{pmatrix}$.

We use $V_{GS} = V_t = 2V$ as our initial guess for Newton-Raphson method. This means that we set $V_3 = 2V$, $V_2 = 0V$, and $V_1 = 0V$ to obtain the initial value for g_m and R_{out}. The method of calculating g_m and R_{out} is shown in Section 3.2.3.

The first Jacobian matrix and RHS column obtained from our program, and the solution for the first iteration is shown below:

Initial Jacobian Matrix	Initial RHS Column	First Vx Solution: Vx(1)	
(0.001, 0) (0, 0) (−0.001, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (1, 0) (0, 0) (−0.001, 0) (0, 0) (0.001, 0) (0, 0) (1, 0) (0, 0) (1, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (1, 0) (0, 0) (0, 0)	(0, 0) (0, 0) (0, 0) (10, 0) (5, 0)	(5, 0) (10, −0) (5, 0) (0, 0) (−4.20671e−16, −0)	N004: (5, 0) N003: (5, 0) N001: (10, −0) N000: (0, 0) N002: (5, 0)

Table 10: Screenshot for the first iteration

Since $V_{GS} = V_4 = 5\text{V} > V_t$, and $V_{DS} = V_1 = 10\text{V} > V_{GS} - V_t$, the MOSFET is in saturation mode.

The second and third iteration have the result:

Jacobian Matrix	RHS Column	Vx Solution	
(0.001, 0) (0, 0) (−0.001, 0) (0, 0) (0, 0) (0, 0) (0.000225, 0) (0, 0) (1, 0) (0, 0) (−0.001, 0) (0, 0) (0.001, 0) (0, 0) (1, 0) (0, 0) (1, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (1, 0) (0, 0) (0, 0)	(0, 0) (0.0525, 0) (0, 0) (10, 0) (5, 0)	(5, 0) (10, −0) (5, 0) (0.05025, −0) (−4.22405e−16, −0)	N004: (5, 0) N003: (5, 0) N001: (10, −0) N000: (0, 0) N002: (5, 0)
Jacobian Matrix	RHS Column	Vx Solution	
(0.001, 0) (0, 0) (−0.001, 0) (0, 0) (0, 0) (0, 0) (0.000225, 0) (0, 0) (1, 0) (0, 0) (−0.001, 0) (0, 0) (0.001, 0) (0, 0) (1, 0) (0, 0) (1, 0) (0, 0) (0, 0) (0, 0) (0, 0) (0, 0) (1, 0) (0, 0) (0, 0)	(0, 0) (0.0525, 0) (0, 0) (10, 0) (5, 0)	(5, 0) (10, −0) (5, 0) (0.05025, −0) (−4.22405e−16, −0)	N004: (5, 0) N003: (5, 0) N001: (10, −0) N000: (0, 0) N002: (5, 0)

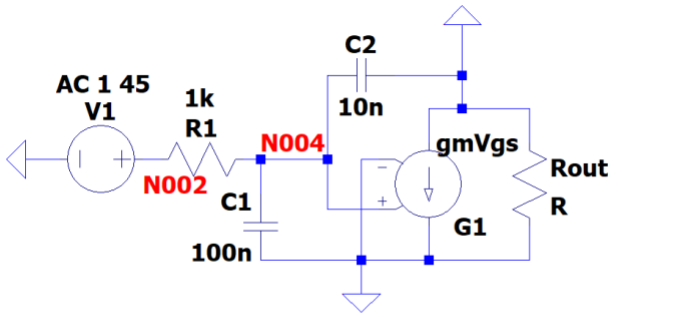
Table 11: Table of iteration process for case 2

The Newton-Raphson process ends after 3 iterations, and the DC operating point of the circuit is obtained. The following table shows the comparison between DC operating voltage obtained by our program and LTSpice.

N004: (5, 0) N003: (5, 0) N001: (10, −0) N000: (0, 0) N002: (5, 0)	--- Operating Point --- V(n001): 10 voltage V(n003): 5 voltage V(n002): 5 voltage V(n004): 5 voltage
Figure 40: DC Operating point found by our program	Figure 41: Operating point found by LTSpice

Table 12: Comparison of DC operating point between our program and LTSpice

Since the DC operating point obtained is correct, we can proceed to AC analysis. The equivalent circuit for AC analysis is:



.model TestN nmos (kp=0.05 vt0=2 lambda=0.01)
.ac dec 10 10 100k

Figure 42: Equivalent circuit for case 2 AC analysis

N004: $\frac{V_4}{R_1} - \frac{V_2}{R_1} + j\omega C_1 \cdot V_4 + j\omega C_2 \cdot V_4 = 0$

N002: $\frac{V_2}{R_1} - \frac{V_4}{R_1} = 0$

$V_2 = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2} i$

The AC analysis is in the range $f = 10Hz$ to $f = 100kHz$. We take $f = 10Hz$ as an example, the conductance matrix and RHS column is:

gmatrix (0.001, 6.9115e-06) (−0.001, 0) (0, 0)	(−0.001, 0) (0.001, 0) (1, 0)	(0, 0) (1, 0) (0, 0)	vicolumn (0, 0) (0, 0) (0.707107, 0.707107)	Vx (0.71196, 0.702186) (0.707107, 0.707107) (4.85316e-06, −4.92071e-06)	N004: (0.71196, 0.702186) N003: (0.707107, 0.707107) N001: (0, 0) N000: (0, 0) N002: (0, 0)

Table 13: AC analysis solution for f=10Hz

We store the voltage value obtained at each frequency step and then use the stored value for the reference node and output node to plot the magnitude and phase versus frequency diagram for the transfer function. In this case, we take N003 as the reference node and N004 as the output node.

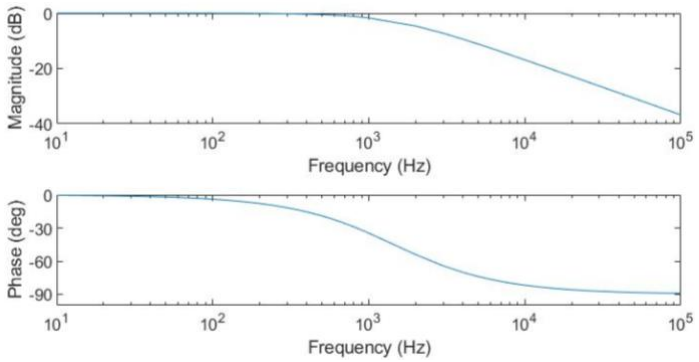


Figure 43: Frequency response plot by our program

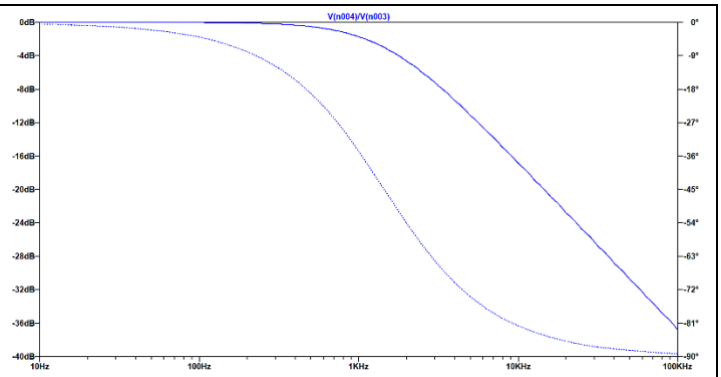


Figure 44: Frequency response plot by LTSpice

Table 14: Comparison of result between our program and LTSpice for case 2

We can observe from the table above that our program works well for finding DC operating point and doing AC analysis for MOSFET in saturation mode.

6.3 Case 3: PNP BJT in Active Mode

Circuit contains PNP BJT.

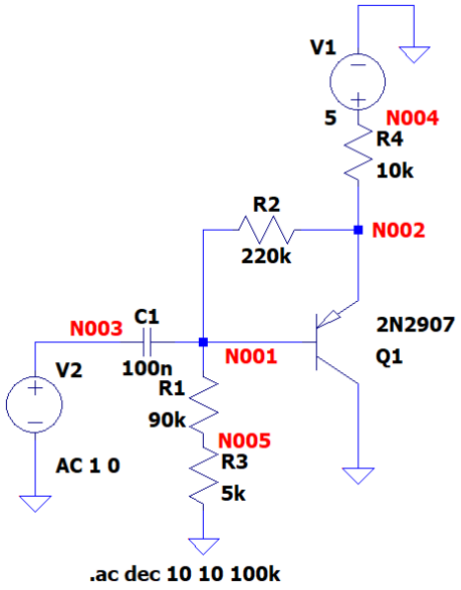


Figure 45: Circuit diagram for case 3

```
V1 N004 0 5
R1 N005 N001 90k
R2 N001 N002 220k
R3 N005 0 5k
R4 N004 N002 10k
V2 N003 0 AC 1 0
C1 N001 N003 100n
Q1 0 N001 N002 PNP
.ac dec 10 10 100k
.end
```

Figure 46: TXT file for circuit simulation for case 3

The detailed steps in testing this case (i.e., writing out the equations and checking for Jacobian matrix and RHS column one by one) are not shown due to the limitation of length. The table below shows the DC operating point comparison between our program and LTSpice.

N004: (5, −0)
N000: (0, 0)
N005: (0.021876, 0)
N001: (0.415645, −0)
N002: (1.03278, −0)
N003: (0, 0)

Figure 47: DC operating point found by our program

```
--- Operating Point ---

V(p001):        5               voltage
V(p002):        0.0221288       voltage
V(n001):        0.420448        voltage
V(n002):        1.05128         voltage
V(n003):        0               voltage
```

Figure 48: DC operating point found by LTSpice

P001 is equivalent to N004; P002 is equivalent to N005

Figure 49: Comparison of DC operating point between our program and LTSpice

From the result, we can tell that the operating point voltages obtained by LTSpice and our program are generally the same. We take N003 as the reference node, N001 as the output node. Then we plot the frequency response.

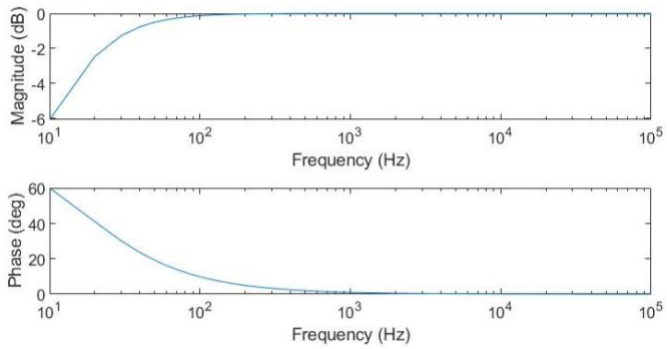


Figure 50: Frequency response plot by our program

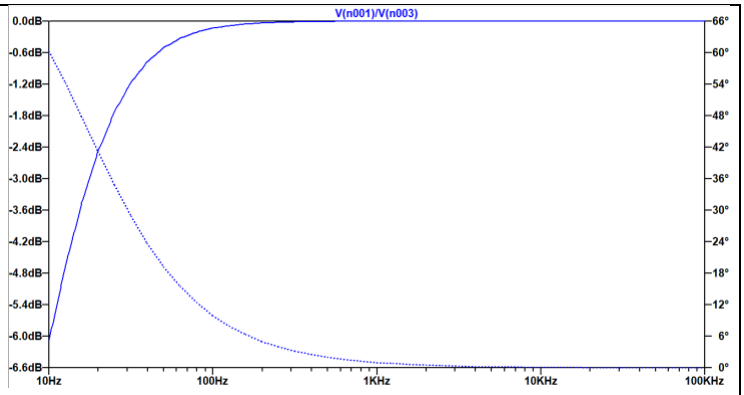


Figure 51: Frequency response plot by LTSpice

Table 15: Comparison of results between our program and LTSpice for case 3

We can observe from the table above that our program produces the correct frequency response plot for PNP BJT as well.

6.4 Case 4: NPN BJT in Reverse-Active Mode

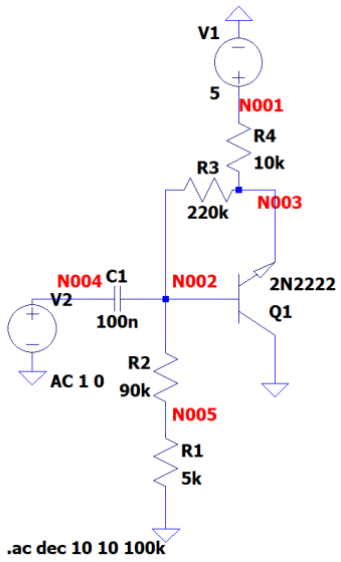


Figure 52: Circuit diagram of case 3

```
V1 N001 0 5
R2 N002 N005 90k
R3 N002 N003 220k
R1 N005 0 5k
R4 N001 N003 10k
V2 N004 0 AC 1 0
C1 N002 N004 100n
Q1 0 N002 N003 NPN
.ac dec 10 10 100k
.end
```

Figure 53: TXT file for circuit simulation

The table below shows the comparison of DC voltages obtained by our program and LTSpice.

```
N001:  (5, 0)
N000:  (0, 0)
N005:  (0.263158, -0)
N003:  (4.31799, 0)
N002:  (0.566906, 0)
N004:  (0, 0)
```

Figure 54: DC operating point obtained by our program

```
--- Operating Point ---

V(n001):        5               voltage
V(n004):        0               voltage
V(n002):        0.568736        voltage
V(n005):        0.0299335       voltage
V(n003):        4.47193         voltage
```

Figure 55: DC operating point obtained by LTSpice

Table 16: Comparison between DC operating point result

We can observe that although there is a slight difference between the results, it is in an acceptable range.

Then we do the AC analysis for the frequency range $f = 10Hz$ to $f = 100kHz$. We choose N004 as the reference node and N002 as the output node. The final graph obtained is shown below:

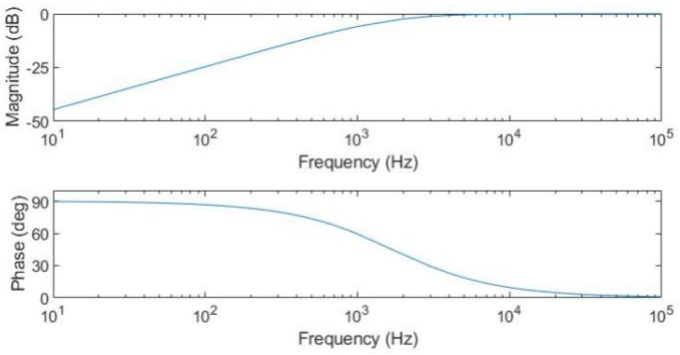


Figure 56: DC operating point obtained by our program

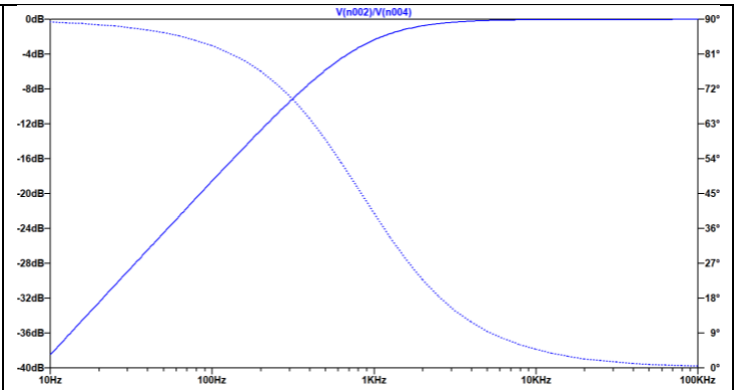


Figure 57: Plot produced by LTSpice for case 3 AC analysis

Table 17: Comparison between AC analysis plot for case 3

From the magnitude and phase plot above, we can observe that our program works for the reverse active mode of BJT.

6.5 Case 5: N-MOSFET in Triode Mode

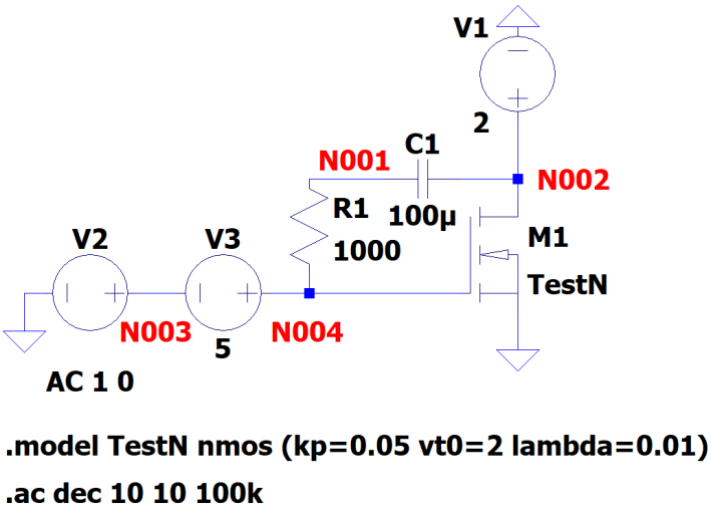


Figure 58: Circuit diagram for case 5

```
V1 N002 0 2
V2 N003 0 AC 1 0
V3 N004 N003 5
C1 N002 N001 100u
R1 N001 N004 1000
M1 N002 N004 0 NMOS
.ac dec 10 10 100k
.end
```

Figure 59 TXT file for case 5

The table below shows the DC operating point comparison between our program and LTSpice.

N002: (2, −0) N000: (0, 0) N003: (0, 0) N004: (5, −0) N001: (5, 0)	--- Operating Point --- V(n002): 2 voltage V(n003): 0 voltage V(n004): 5 voltage V(n001): 5 voltage
Figure 60: DC operating point found by our program	Figure 61: DC operating point found by LTSpice

Table 18: Comparison of DC operating point between our program and LTSpice

From the result, we can tell that the DC operating point voltages obtained by LTSpice and our program are identical.

We take N003 as the reference node, N001 as the output node and do AC analysis. Then we plot the frequency response.

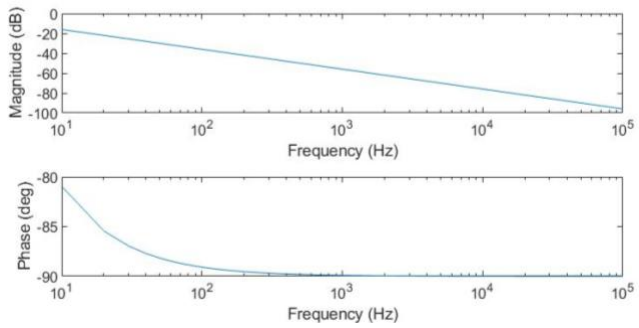


Figure 62: Frequency response plot by our program

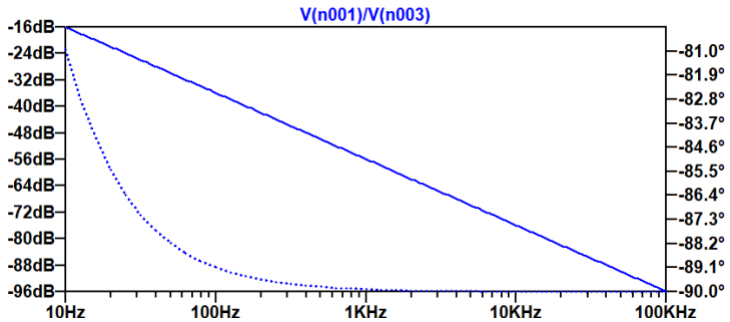


Figure 63: Frequency response plot by LTSpice

Table 19: Comparison of results between our program and LTSpice for case 3

We can observe from the table above that our program produces correct frequency response plot for MOSFET in triode mode as well.

7 Evaluation

We need to compare our outcome against the functional and non-functional requirements listed in the SRS, to evaluate if our program has met these requirements.

7.1 Evaluation on Functional Requirements

Read a netlist file and parse the data correctly – Fully met
Netlist files can be correctly read. Units are converted correctly, and component data are stored in the correct vector. Details are shown in Section 4.3.

Convert the components into their equivalent models to be used in AC and DC analysis – Fully met
Section 4.4 and Section 4.5 show the implementation of short-circuiting and open-circuiting. The detailed tests in the sections show that the node objects are correctly changed when required. Section 4.6 indicates that objects of equivalent models are correctly created.

Perform DC analysis to find the operating point – Fully met
Our program can calculate the correct DC operating point using Newton-Raphson method. Using our research findings described in Section 3.2, we have chosen the correct formulae and implemented the Newton Raphson method properly. We have tested each function as well as the overall DC analysis program. Testing details are in Section 4.8 ~ 4.11.

Perform AC analysis to find the transfer function – Fully met
We have used the MNA (Modified Nodal Analysis) method to implement AC analysis. Using the research findings in Section 3.1, we have successfully implemented and tested our program.

Select the correct reference and output nodes. The output should contain a magnitude-frequency graph and a phase-frequency graph – Fully met
We have decided to let the user enter their desired reference and output nodes. By comparing the result generated by our program and LTSpice in the previous 5 examples, we can conclude that the magnitude and phase plots produced are smooth and accurate.

Extra requirement: the program should determine the mode of the diode, BJT and MOSFET， and it should work correctly for all modes – Partially met (BJT saturation mode need to be improved)
Our program manages to determine different modes of non-linear devices (Section 4.2). After doing a few tests, we observe that if the input circuit contains BJT in saturation mode, the obtained DC operating point doesn't match the data produced from LTSpice. Findings show that it may be due to the use of different formulae to LTSpice. For details, see Section 8.1.

Despite the shortcoming in BJT saturation mode, tests in Section 6 have shown that the circuit simulator can work correctly for all other modes:

- Test 1: NPN BJT in active mode
- Test 2: N-MOSFET in saturation mode
- Test 3: PNP BJT in active mode
- Test 4: NPN BJT in reverse-active mode
- Test 5: N-MOSFET in triode mode

By using a range of different data to test each situation, we have ensured that our program can work correctly in most cases.

7.2 Evaluation on Non-functional Requirements

Performance: the execution time of the program should be as short as possible – Mostly met

In order to assess the performance of our program further, we need to measure the running time, as well as energy consumption for each testing case. We use 30W as the average power consumption of the computer and calculate energy consumption using the multiplication of running time and average power consumption.

	Number of Components	Running Time	Average Power Consumption of Computer	Energy Consumption of Program
Case 1	8	7.702s	30W	231.06J
Case 2	7	4.031s	30W	120.93J
Case 3	7	7.648s	30W	229.44J
Case 4	8	7.542s	30W	226.26J
Case 5	6	4.246s	30W	127.38J

Table 20: Table of running assessment

We can observe that the running time and energy consumption of the program have little relation with the number of components or the number of nodes. However, we can see that generally analyzing circuits containing BJT requires more running time than circuits containing MOSFET. This is because MOSFET converging to its DC operating point faster than BJT.

Overall, the program can finish the analysis in a reasonable amount of processing time.

Portability and compatibility: The program should be portable, so moving from one OS to another OS doesn't create any problem – Fully met

We have tested running the program using the 3 most popular operating systems: Windows, macOS and Linux. And we have proven that our program works correctly on these operating systems.

8 Future Work

Besides the advantages of our project, there are some shortcomings and areas for further improvement.

8.1 Not Working Well for BJT Saturation Mode

After testing for a few examples, we observe that if the input circuit contains BJT in saturation mode, the DC operating point obtained is incorrect. The following example is a circuit containing BJT in saturation mode.

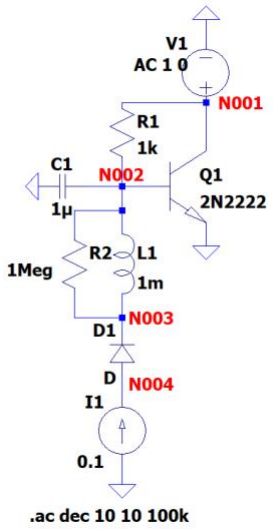


Figure 64: Circuit diagram for saturation case

```
C1 N002 0 1u
V1 N001 0 AC 1 0
Q1 N001 N002 0 NPN
R1 N001 N002 1k
L1 N002 N003 1m
D1 N004 N003
I1 0 N004 0.1
R2 N002 N003 1Meg
.ac dec 10 10 100k
.end
```

Figure 65 TXT file for saturation case

The comparison of DC operating point found by our program and LTSpice is shown below:

$$\begin{aligned} &\mathrm{N002}: \ (0.741527, -0) \\ &\mathrm{N000}: \ (0, 0) \\ &\mathrm{N001}: \ (0, 0) \\ &\mathrm{N003}: \ (0.741527, -0) \\ &\mathrm{N004}: \ (1.50202, 0) \end{aligned}$$

Figure 66: DC operating point found by our program

```
--- Operating Point ---

V(n002):        1.79916         voltage
V(n001):        0               voltage
V(n003):        1.79926         voltage
V(n004):        2.5735          voltage
Ic(Q1):         -0.0515515      device_current
Ib(Q1):         0.0982008       device_current
Ie(Q1):         -0.0466494      device_current
```

Figure 67: DC operating point produced by LTSpice

From the result, we can see that our project works well for analyzing the diode because N004 and N003 have a voltage difference of about $0.7V$. However, the voltage at the base of the BJT is wrong. We have checked the Jacobian matrix system built with Equation set 11, but we didn't find anything wrong with it.

A possible reason for the error: After substituting $V_{BE} = V_2 = 1.79916V$, $V_{BC} = V_{21} = 1.79916V$ into Equation set 10, we found that $I_C = -5.9909 \times 10^{16}A$, which is not equal to what is shown by LTSpice $I_C = -0.0515515A$. Since our calculation by hand doesn't match the calculation in LTSpice, it is reasonable to deduce that LTSpice might use a different set of equations to calculate I_C, I_B, and I_E, which could be the reason for this problem. More work can be done in the future to resolve this problem.

8.2 Require Opening of 2 Software

Currently, our program requires running the code in visual studio and then open MATLAB to produce the graphs, this is time-consuming and indirect. This issue can be improved by writing code that opens MATLAB automatically. An alternative would be to use a library that allows the program to generate graphs directly from C++ code.

8.3 Shorten Execution Time

Despite our program runs in a reasonable amount of time, there is still room for further execution time reduction. This could possibly be achieved by choosing more suitable initial guess data for the Newton Raphson iteration or using other numerical methods other than Newton Raphson approximation.

9 Project Planning and Management

9.1 Milestone

Project start date: 10 May 2021

Milestones	Planned deadline	Days (planned)	Actual finish date	Days used (actual)
Initial research & design	13 May	4	12 May	3
AC analysis code writing	23 May	10	18 May	6
DC analysis code writing	4 June	12	31 May	13
Finish & debug the whole program	7 June	3	4 June	4
Finish writing the report	12 June	5	12 June	8
Video making & editing	12 June	5	13 June	4

9.2 Meeting Structure

Date	Meeting Time	Brief
10 May	9:30 – 11:00	Discuss understanding of the project & start to research relevant methods and formulae
11 May	9:30 – 11:00	Discuss problem outline and breakdown
12 May	9:30 – 11:30	Allocate coding tasks and start programming – write classes for the components
13 May	11:00 – 11:20	Consultation
	11:30 – 12:30	Discuss & allocate coding tasks – small sig. equiv. conversion & continue writing classes
14 May	9:30 – 11:00	Discuss research result on MNA implementation & choose matrix manipulation library
15 May	No meeting	N/A
16 May	No meeting	N/A
17 May	9:30 – 11:00	Confirm details on MNA method & start considering cases in different modes (e.g., saturation, reverse, cut-off)
18 May	11:00 – 11:30	Consultation
	11:30 – 12:30	Finalize AC analysis
19 May	10:30 – 11:00	Consultation
	11:00 – 12:00	Discuss and research Newton Raphson method & Jacobian matrix construction
20 May	9:30 – 11:00	Continue from the last meeting

21 May	9:30 – 11:00	Discuss changes in classes to satisfy the new equivalent model
22 May	No meeting	N/A
23 May	No meeting	N/A
24 May	9:30 – 11:00	Confirm Newton Raphson code implementation method
25 May	9:30 – 11:00	Discuss the problem with nodes and find solutions to fix it
26 May	No meeting	N/A
27 May	9:30 – 11:00	Discuss what components need to become open/short circuit
28 May	9:30 – 11:00	Discuss & confirm further details
29 May	9:30 – 11:00	Consider how the data is outputted & start to learn MATLAB
30 May	9:30 – 11:00	Start to debug the whole program
31 May	9:30 – 11:00	Decide what test data files to use & start to debug the logic and functional errors
1 June	No meeting	N/A
2 June	No meeting	N/A
3 June	9:30 – 11:00	Decide report structure & format
4 June	9:30 – 11:00	Further discussion on report structure & flowchart modification
5 June	9:30 – 12:30	Work together on the report
6 June	9:30 – 12:30	Work together on the report
7 June	8:30 – 15:00	Divide report sections and do each part separately
8 June	10:30 – 11:30 12:00 – 14:00	Discuss & confirm video content & continue doing the report
	11:30 – 12:00	Consultation
9 June	9:30 – 13:00	Further discussion on report
10 June	9:30 – 13:00	Discuss and make video ppt; Proof-read report
11 June	9:30 – 13:00	Record video
12 June	10:00 – 14:30	Proof-read report and submit source code

9.3 Gantt chart

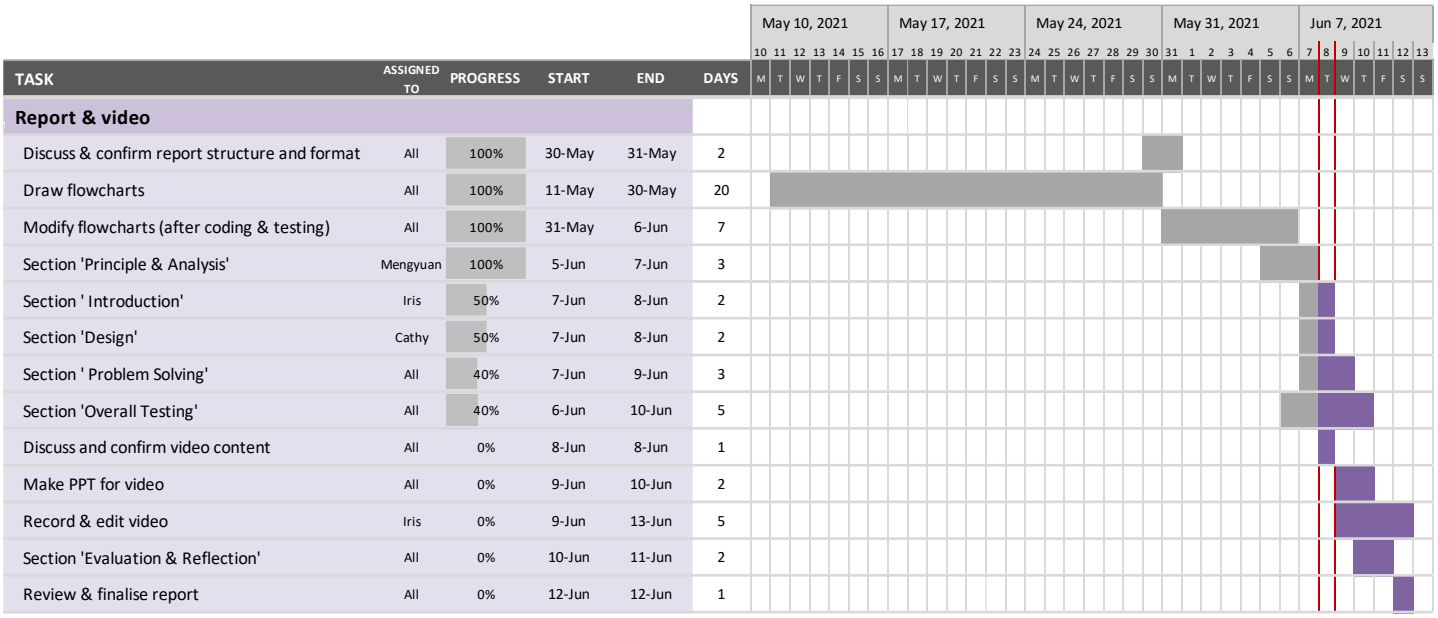
This chart is put into the report on 8th June, so there are still some uncompleted tasks as shown in the end of the diagram.

Project Gantt Chart

Project Start: 10/05/2021

Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	END	DAYS
Design & Research					
Read & understand the project	All	100%	10-May	11-May	2
Research relevant formulae	All	100%	10-May	12-May	3
Breakdown the problem into tasks	All	100%	11-May	12-May	2
Research MNA implementation (AC)	All	100%	14-May	15-May	2
Research Newton Raphson method (DC)	All	100%	17-May	21-May	5
Research formulae for I and g (DC)	Mengyuan	100%	23-May	25-May	3
Further Reseach on PNP & PMOS	All	100%	5-Jun	6-Jun	2
AC Analysis code writing					
Read file & data parsing	Iris	100%	11-May	12-May	2
Write classes for RLC	Mengyuan	100%	12-May	13-May	2
Write classes for diode, BJT & MOSFET	Cathy	100%	12-May	13-May	2
Convert to small signal equivalent	Cathy	100%	13-May	14-May	2
Write classes for I and V sources	Iris	100%	13-May	13-May	1
Make test data files for testing	Iris	100%	14-May	14-May	1
MNA matrix implementation	Mengyuan	100%	14-May	17-May	4
Write classes for non-linear equiv R	Cathy	100%	16-May	18-May	3
Determine mode	Iris	100%	18-May	18-May	1
DC Analysis code writing					
Newton Raphson iteration loop	Iris	100%	19-May	19-May	1
Write classes for non-linear equiv G	Mengyuan	100%	22-May	23-May	2
Newton Raphson matrix manipulation	Mengyuan	100%	24-May	25-May	2
Write classes for non-linear equiv I source	Iris	100%	25-May	26-May	2
Modify classes for non-linear equiv R	Cathy	100%	25-May	26-May	2
Modify classes for non-linear equiv G	Mengyuan	100%	25-May	26-May	2
Write functions in classes to get the constants	Cathy	100%	25-May	25-May	1
Build Jacobian matrix	Mengyuan	100%	26-May	28-May	3
Write new non-linear equiv conversion (DC&AC)	Iris	100%	27-May	28-May	2
Short circuit	Cathy	100%	27-May	28-May	2
Build V/I source column	Mengyuan	100%	29-May	29-May	1
Format and write output data into file	Iris	100%	29-May	29-May	1
Write node class	Cathy	100%	29-May	29-May	1
Modify accordingly using the node class	All	100%	30-May	30-May	1
Open circuit	Iris	100%	30-May	30-May	1
Debug whole program (syntax & runtime errors)	All	100%	30-May	31-May	2
Make test data files for testing	Mengyuan	100%	31-May	31-May	1
Debug whole program (logic & functional errors)	All	100%	31-May	4-Jun	5



TASK	ASSIGNED TO	PROGRESS	START	END	DAYS
Report & video					
Discuss & confirm report structure and format	All	100%	30-May	31-May	2
Draw flowcharts	All	100%	11-May	30-May	20
Modify flowcharts (after coding & testing)	All	100%	31-May	6-Jun	7
Section 'Principle & Analysis'	Mengyuan	100%	5-Jun	7-Jun	3
Section ' Introduction'	Iris	50%	7-Jun	8-Jun	2
Section 'Design'	Cathy	50%	7-Jun	8-Jun	2
Section ' Problem Solving'	All	40%	7-Jun	9-Jun	3
Section 'Overall Testing'	All	40%	6-Jun	10-Jun	5
Discuss and confirm video content	All	0%	8-Jun	8-Jun	1
Make PPT for video	All	0%	9-Jun	10-Jun	2
Record & edit video	Iris	0%	9-Jun	13-Jun	5
Section 'Evaluation & Reflection'	All	0%	10-Jun	11-Jun	2
Review & finalise report	All	0%	12-Jun	12-Jun	1

10 Reflection

During the project, despite the existence of time difference, our group has overcome the difficulty and cooperated closely. We have group meeting time every day. During the meeting, we update our progress, analyze the tasks together, and discuss possible solutions to the problems met. We also record down important points mentioned during the meeting and our division of works (shown in Section 9). This allows us to work individually and effectively in our own time. Teams was used as our main communication tool. Coding source files are shared on GITHUB to keep our project organized. There were some inconveniences as we were only able to work together online, lack of face-to-face discussion reduced working speed.

Iris: In this project, I corporate with my teammates for more than a month and produced a completed circuit simulator. This helped me to improve my teamwork skill as well as communication skills. I worked hard and finished my work, but I found that my research skill is not good enough and I learnt a lot about researching from my teammates.

Mengyuan: I have deepened my understandings of circuit analysis from researching relevant subjects. Moreover, this project allows me to enhance my coding skills in C++ and familiarize myself with MATLAB. However, sometimes I found myself quite unclear when communicating academic ideas. I will work on this area in the future.

Cathy: Through this project, I have learnt to listen and work collaboratively as a team, accepting each other's opinions and expressing my thoughts. It has also given me an insight into how a big project goes from design to completion. However, for a project like this, programming skills and circuit analysis understanding are both very important. The lack of understanding of circuit analysis was a drawback for me and this is what I need to improve on.

11 References

[1] Philips Semiconductors, “NPN switching transistors,” 2N2222; 2N222A datasheet, 29 May 1997.

[2] Philips Semiconductors, “PNP switching transistors,” 2N2907; 2N2907A datasheet, 30 May 1994.

[3] Eigen.tuxfamily.org. (2021). *Eigen* [Online]. Available: https://eigen.tuxfamily.org/index.php?title=Main_Page.

[4] C. Ho, A.Ruehli and P. Brennan, “The modified nodal approach to network analysis,” IEEE Transactions on Circuits and Systems, vol. 22, no. 6, pp. 504-509, Jun 1975.

[5] A. J. Conejo and B. Luis, “Appendix A: Solving Systems of Nonlinear Equations,” in *Power System Operation*, New York: Springer, 2018, pp. 271-279.

[6] S. Jahn, M. Margraf, V. Habchi and R. Jacob, “Non-linear DC Analysis,” Dec. 2007. [Online]. Available: http://qucs.sourceforge.net/tech/node16.html.

[7] A. Marti, J.L. Balenzategui and R. F. Reyna, “Photon recycling and Shockley’s diode equation,” *Journal of Applied Physics*, vol. 82, no.8, pp. 4067-4075, 1997.

[8] C.G. Sodini, “Lecture 18: The Bipolar Junction Transistor (II),” Apr. 2007. [Online]. Available: http://web.mit.edu/6.012/www/SP07-L18.pdf.

[9] A. S. Sedra and S. C. Kenneth, “Bipolar Junction Transistors (BJTs),” in Microelectronic Circuits, J. Lee, Ed. New York: Oxford, 2014, pp. 316-318.

[10] A. Agarwal and J. H. Lang, “The MOSFET Amplifier,” in *Foundations of Analog and Digital Electronic Circuits*, D. E. Penrose, Ed. Amsterdam: Elsevier, 2005, pp. 382-388.

[11] A. R. Newton, “Computer Analysis of Electrical Circuits,” Sep. 1998. [Online]. Available: https://people.eecs.berkeley.edu/~newton/Classes/EE219fa98/ee219a3_2/ee219a3_2.pdf.